2015-01-23

Computer Vision Aiding Smartphone Sensors for Indoor Location Applications

Kazemipur, Bashir

UNIVERSITY OF CALGARY

Computer Vision Aiding Smartphone Sensors for 3D Indoor Location Applications

By

Bashir Kazemipur

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF GEOMATICS ENGINEERING

CALGARY, ALBERTA

January 2015

Abstract

Modern mobile phones are powerful processing devices with a host of onboard technologies of interest to navigation system designers. In the absence of Global Navigation Satellite System (GNSS) information, the accelerometers and gyroscopes within a smartphone can be used to provide a relative navigation solution. However, these micro-electro-mechanical systems (MEMS) based sensors suffer from the effects of various errors which cause the inertial-only solution to deteriorate rapidly. As such, there is a need to constrain the inertial positioning solution when long-term navigation is needed. GNSS positions and velocities, and WiFi positions when available, are the most important forms of updates available for the inertial solution. However, updates from these two sources depend on external signals and infrastructure that may not always be available. One attractive source of updates is through the use of a vision sensor.

This work describes the development of a vision-based module that determines the device heading misalignment and context based on a sequence of images captured from the device camera. The vision aiding module checks for static periods and calculates the device heading misalignment when in motion. Context classification is assessed for five common use cases: (1) fidgeting the phone while standing still ("fidgeting" context), (2) phone on ear on one floor ("single floor calling" context), (3) phone on ear on stairs ("stairs calling" context), (4) phone in hand on a single floor ("single floor texting" context), and (5) phone in hand on stairs ("stairs texting" context). The module was tested using real-time video and inertial data collected using a Samsung Galaxy S3 smartphone running the Android 4.0 operating system. The results show successful detection of the aforementioned use cases and accurate device angles. Integration of the vision aiding module with a pedestrian dead reckoning (PDR) system shows improvements to the position solution.

Acknowledgements

I would like to thank Dr. Naser El-Sheimy for supporting me as a graduate student and providing me with the opportunity to pursue studies with the Mobile Multi-Sensors System (MMSS) Research Group. I would like to thank Dr. Zainab Syed and Dr. Jacques Georgy as this work would not have been possible without their knowledge and problem-solving skills, as well as their continual guidance, feedback, and mentorship. I would also like to thank Dr. Chris Goodall for continually presenting me with challenging projects and motivating me with his own passion towards technology and entrepreneurship.

I would like to thank my circle of friends for their constant support as well as my officemates, especially Dave and James, whose sense of humor created the best work environment I could ask for. They were all instrumental to the completion of this work.

Lastly, I would like to thank my parents Abdie and Marzieh and my brother Shakeeb. It would take many times the length of this entire work to fully express what I am grateful to them for. Suffice it to say that I would not be where or who I am today without everything that they have provided me with.

Table of Contents

List of Tables

List of Figures and Illustrations

List of Symbols

Symbol	Definition
$\tilde{\boldsymbol{f}}^{\boldsymbol{b}}$	reported accelerometer measurement [m/s^2]
$\boldsymbol{f}^{\boldsymbol{b}}$	true acceleration [m/s^2]
$\boldsymbol{b}_{\boldsymbol{a}}$	accelerometer bias vector [m/s^2]
$\boldsymbol{S}_{\boldsymbol{1}}$	accelerometer scale factor error
$\boldsymbol{S}_{\boldsymbol{2}}$	accelerometer scale factor non-linearity error
$\boldsymbol{N}_{\boldsymbol{a}}$	accelerometer axis non-orthogonality error
$\boldsymbol{\delta g}$	difference between the true and theoretical gravity value
$\boldsymbol{\varepsilon}_{\boldsymbol{a}}$	accelerometer noise [m/s^2]
$\boldsymbol{e}_{\boldsymbol{a}}$	cumulative magnitude of all uncompensated accelerometer errors [m/s^2]
$\tilde{\boldsymbol{\omega}}_{\boldsymbol{ib}}^{\boldsymbol{b}}$	reported gyroscope measurement [deg/h]
$\boldsymbol{\omega}_{\boldsymbol{ib}}^{\boldsymbol{b}}$	true rotation rate [deg/h]
$\boldsymbol{b}_{\boldsymbol{g}}$	gyroscope bias vector [deg/h]
$\boldsymbol{S}_{\boldsymbol{g}}$	gyroscope scale factor error
$\boldsymbol{N}_{\boldsymbol{g}}$	gyroscope axis non-orthogonality error
$\boldsymbol{\varepsilon}_{\boldsymbol{g}}$	gyroscope noise [deg/h]
$\tilde{\boldsymbol{\omega}}^{\boldsymbol{b}}$	reported gyroscope measurement [deg/h]
$\boldsymbol{\omega}^{\boldsymbol{b}}$	true rotation rate [deg/h]
$\boldsymbol{e}_{\boldsymbol{g}}$	cumulative magnitude of all uncompensated gyroscope errors [deg/h]
$\tilde{\boldsymbol{\theta}}$	total rotation angle obtained using the reported gyroscope measurement [deg]
$\boldsymbol{\theta}$	true total rotation angle [deg]
$\boldsymbol{\delta\theta}$	rotation angle error [deg]
$\boldsymbol{G}_{\boldsymbol{k}}$	k^{th} grayscale image in sequence
$\boldsymbol{N}_{\boldsymbol{x}}$	image width
$\boldsymbol{N}_{\boldsymbol{y}}$	image height
$\boldsymbol{I}$	intensity
$\boldsymbol{I}_{\boldsymbol{max}}$	maximum intensity

$\boldsymbol{p}$	normalized histogram of image
$\boldsymbol{p_n}$	fraction of pixels in image with intensity *n*
$\boldsymbol{g}$	histogram equalized image
$\boldsymbol{K}$	convolution kernel
$\boldsymbol{x}$	distance from origin in the horizontal image axis
$\boldsymbol{y}$	distance from origin in the vertical image axis
$\boldsymbol{\sigma}$	standard deviation
$\boldsymbol{\sigma^2}$	variance
$\boldsymbol{\mu}$	mean
$\boldsymbol{f}$	source image
$\boldsymbol{G_x}$	image gradient along x axis
$\boldsymbol{G_y}$	image gradient along y axis
$\boldsymbol{G}$	magnitude of image gradient
$\boldsymbol{D}$	aggregated image gradient
$\boldsymbol{D_{norm}}$	normalized aggregated image gradient
$\boldsymbol{C_1}$	first stabilizing coefficient
$\boldsymbol{C_2}$	second stabilizing coefficient
$\boldsymbol{t}$	time
$\boldsymbol{u_x}$	optical flow in the x axis
$\boldsymbol{u_y}$	optical flow in the y axis
$\boldsymbol{A}$	symmetric matrix
$\boldsymbol{b}$	vector
$\boldsymbol{c}$	scalar
$\boldsymbol{d}$	global displacement
$\boldsymbol{dx_{agg,k}}$	x-component of aggregate translation for image *k*
$\boldsymbol{dy_{agg,k}}$	y-component of aggregate translation for image *k*
$\boldsymbol{dx_{agg,k}^{binary}}$	binary thresholded x-component of aggregate translation for image *k*
$\boldsymbol{dy_{agg,k}^{binary}}$	binary thresholded y-component of aggregate translation for image *k*
$\boldsymbol{D_{x,k}}$	differenced binary thresholded x-component of aggregate translation for image *k*
$\boldsymbol{D_{y,k}}$	differenced binary thresholded y-component of aggregate translation for image *k*

$\boldsymbol{M}$	magnitude spectrum of signal
$\boldsymbol{Y}$	Fourier transform of signal
$\boldsymbol{\omega_N}$	N^{th} root of unity
$\boldsymbol{f_{dangling}}$	dangling frequency
$\boldsymbol{\theta_k}$	device angle
$\boldsymbol{dx_{agg,k}^{floor}}$	x-component of aggregate translation for the floor region of image k
$\boldsymbol{dy_{agg,k}^{floor}}$	y-component of aggregate translation for the floor region of image k
$\boldsymbol{dx_{cum}}$	cumulative x-component of aggregate image translation
$\boldsymbol{dy_{cum}}$	cumulative y-component of aggregate image translation
$\boldsymbol{R}$	resultant vector length
$\boldsymbol{N}$	number of samples
$\boldsymbol{V}$	circular variance of device angle
$\boldsymbol{v}$	circular standard deviation of device angle

List of Abbreviations

Abbreviation	Definition
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
MEMS	Micro-electro-mechanical Systems
NHC	Negative Half-Cycle
PHC	Positive Half-Cycle
PVA	Position, Velocity, Attitude
RANSAC	Random Sample Consensus
SFM	Structure From Motion
SIFT	Scale Invariant Feature Transform
SLAM	Simultaneous Localization and Mapping
SURF	Speeded Up Robust Features

Chapter 1: Introduction

A logical entry point for this work involves a description of the motivation behind it and the problem it attempts to address, a list of objectives to clearly define the scope of the research, as well as a general outline of the individual chapters.

1.1 Motivation and Problem Statement

An inertial navigation system (INS) is a means of using inertial sensors attached to a moving platform to provide a navigation solution in the form of the platform's position, velocity, and attitude (PVA) as it moves through space [1]. The two major components of an INS are the inertial measurement unit (IMU) and the mechanization module [1]. The IMU consists of three accelerometers and three gyroscopes arranged in a mutually orthogonal fashion which measure the body's three-dimensional (3-D) translational and rotational motion, respectively [1]. When the body undergoes motion, the mechanization module (the mathematical core) translates the accelerometer and gyroscope signals into a PVA solution. In an ideal world, twice-integrating the transformed accelerations would give an accurate position and velocity solution. With practically realizable systems, we run into two major problems that are further exacerbated by the fact that the navigation devices forming the subject of this work are portable and handheld.

The first problem stems from the fact that manufactured inertial sensors suffer from various errors that arise from their underlying architecture. Consumer-grade portable handheld devices (e.g. smartphones) aim to provide a competitive price-point in the market so it follows that the sensors used in such devices are designed to be as small and inexpensive as possible. It is these requirements that make micro-electro-mechanical systems (MEMS) an attractive design choice. Since the magnitudes of the error components are inversely proportional to the cost of the sensor

itself [2], readings from the low-cost MEMS accelerometers and gyroscopes used in these devices suffer from a multitude of errors. These errors fall into the two major categories of systematic and stochastic errors [1].

Systematic errors are those that can be removed through a laboratory calibration [2]**.** Stochastic errors cannot be removed through calibration and must be accounted for using mathematical models [2]. Uncompensated errors in the accelerometer readings result in a velocity error that increases linearly with time and a corresponding position error that increases quadratically with time [1]. Uncompensated errors in the gyroscope readings result in a quadratically increasing velocity error and cubically increasing position error [1].

The second, and much more significant, problem arises from the fact that a portable handheld navigation device is inherently untethered and free to be used in any three-dimensional orientation. An incorrect device misalignment has a similar effect to an uncompensated gyroscope error in that both lead to an incorrect projection of the accelerometer measurements. As such, an incorrect device misalignment leads to a velocity error that increases quadratically with time and a corresponding position error that increases cubically with time.

Unbounded exponential error growth (or *drift*) makes accurate long-term inertial-only navigation a challenging problem that necessitates the use of additional sources of information that can act as constraints. One particularly attractive means of providing such constraints is through the use of a vision sensor.

1.2 Research Objectives

The objectives of this work are three-fold:

1. To develop a software module that that can calculate the angle of a handheld portable navigation device with respect to the direction of its motion using a series of images captured by the device's built-in camera.
2. To develop a software module that can use parameters extracted from the aforementioned images to perform context classification. The classified contexts will reflect whether the device is static or in motion ("static detection"), distinguish between meaningful and non-meaningful motion ("fidgeting detection"), and determine whether the device is undergoing motion on a single floor in a texting or calling use case ("single floor texting use case" or "single floor calling use case", respectively) or undergoing a relative change in height in either use case ("stairs texting use case" or "stairs calling use case", respectively).
3. To encapsulate the aforementioned modules into a single entity that can be used to update an inertial navigation solution for hand-held devices in real-time.

1.3 Thesis Outline

This thesis includes seven chapters. Chapter **one** provides an overview of the motivation behind this work and the problem being addressed. The research objectives are stated and an outline is given of the chapters that follow.

Chapter **two** is comprised of a literature review describing the current role of computer vision within the domain of pedestrian navigation as well as the current achieved accuracy of vision-based and vision-augmented systems.

Chapter **three** begins with a high-level overview of the two sub-modules that comprise the vision aiding module developed in this work. This chapter focuses on providing the mathematical background and methodology of the first sub-module whose three main tasks are to perform availability determination, static detection, and device angle calculation.

Chapter **four** describes the methodology of the second sub-module whose task is to perform context classification. Using the inputs of device angle and externally supplied pitch and roll, this sub-module distinguishes whether the device is fidgeting, travelling up/down stairs in the texting use case, or travelling up/down stairs in the calling use case.

Chapter **five** describes the trajectories, hardware, and software used in the testing of the device angle sub-module. Results are presented for the three tasks described in chapter three.

Chapter **six** describes the trajectories, hardware, and software used in the testing of the context classification sub-module. Results are presented for the five contexts described in chapter four.

Chapter **seven** describes the results of integrating the complete vision aiding module with a smartphone based pedestrian dead reckoning system.

Chapter **eight** summarizes the findings of this work and provides suggestions as to the next steps to be taken in this line of research.

Chapter 2: Background

Understanding the context of this work necessitates a brief description of inertial navigation, pedestrian dead reckoning, and the manner in which vision systems have traditionally been used in the realm of navigation. This chapter begins with a succinct overview of the how inertial sensors are used to form a navigation solution and the mathematical basis for two major problems present in traditional inertial navigation. The discussion proceeds with the advantages offered by pedestrian dead reckoning (PDR) and the mathematical basis by which the vision aiding module will aid the PDR solution. The chapter concludes with a brief overview of the literature available on vision based navigation using examples of portable systems where available. The aim is to showcase the breadth in which computer vision can be used for navigation purposes and to place this work in relation to its relevant category.

2.1 Background to Inertial Navigation

As mentioned in the previous chapter, the two major components of an INS are the IMU and the mechanization module. A simplified overview of INS mechanization is as follows and is summarized in Fig. 2.1 [1]:

1. The body undergoes motion. The body's translational motion is detected and quantified by the accelerometers while the body's rotational motion is detected and quantified by the gyroscopes.
2. The rotational information from the gyroscopes is used to transform the translational accelerometer data into the desired navigational reference frame. The rotational information from the gyroscopes also updates the attitude solution.
3. The transformed accelerations are compensated for gravity and the Coriolis effect.

4. The compensated accelerations are integrated once to provide a velocity solution and integrated again to provide a position solution in the navigational reference frame.

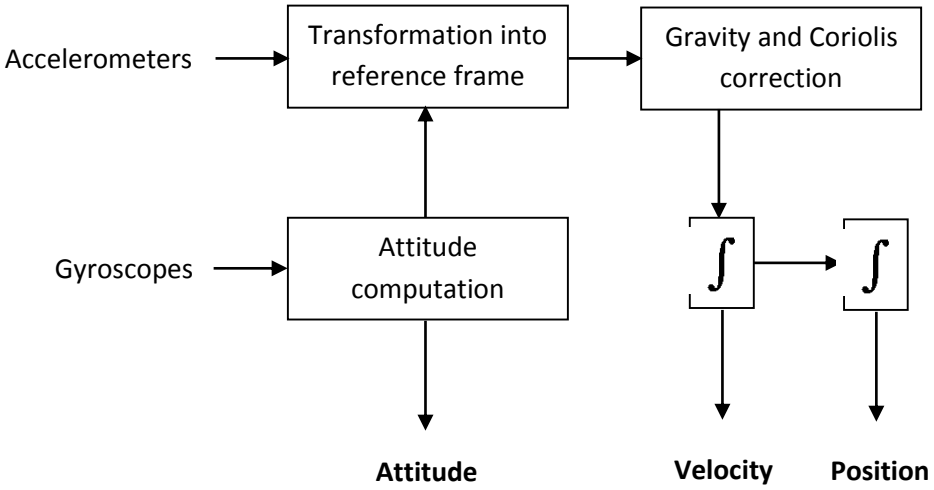


Fig 2.1. Simplified overview of INS mechanization.

In an ideal world, with ideal sensors, the discussion would end here and twice-integrating the transformed accelerations would give an accurate position and velocity solution. In the real world, with practically realizable systems, we run into two major problems that are further exacerbated by the fact that the navigation devices forming the subject of this work are portable and handheld.

The first problem stems from the fact that manufactured inertial sensors suffer from various errors that arise from their underlying architecture [1]. Consumer-grade portable handheld devices (e.g. smartphones) aim to provide a competitive price-point in the market so it follows that the sensors used in such devices are as small and low-cost as possible. It is these requirements that make micro-electro-mechanical systems (MEMS) an attractive design choice. Since the magnitudes of the error components are inversely proportional to the cost of the sensor itself [2], readings from the low-cost MEMS accelerometers and gyroscopes used in these devices suffer

from a multitude of errors. These errors fall into the two major categories of systematic and stochastic errors [1].

Systematic errors are those that can be removed through a laboratory calibration [1]**.** Examples of such errors are the bias offset (the sensors output in the presence of zero input), scale factor error (the degree to which the output does not follow a unity slope linear relation to the input), non-linearity (the degree to which the output does not follow a linear relation to the input), scale factor sign asymmetry (the change in scale factor for positive and negative inputs), dead zone (the opposite of bias offset--the region in which the sensor reports no output in the presence of non-zero input), quantization error (the loss of precision that occurs in any analog-to-digital conversion), non-orthogonality (the degree to which the sensor axes are not perfectly orthogonal to each other), and misalignment error (the degree to which the sensor axes do not line up with the axis of the body or platform that the sensor is mounted on) [2].

Stochastic errors cannot be removed through calibration and must be accounted for using mathematical models [1]. Examples of stochastic errors include the run-to-run bias offset (the change in the aforementioned bias offset error every time the sensor is powered on), bias drift (the random change in the sensor bias with time), scale factor instability (random change in the aforementioned scale factor every time the sensor is powered on), and white noise (an uncorrelated noise distributed across the sensor's entire power spectrum) [2].

Taking into account the aforementioned errors in the sensor readings, the three-element measurement vector reported by a three-axis accelerometer can be modeled by the following equation [1]:

$$\tilde{f}^b = f^b + b_a + S_1 f^b + S_2 f^b + N_a f^b + \delta g + \varepsilon_a \quad (1)$$

where

$\tilde{\boldsymbol{f}}^{\boldsymbol{b}}$ is the reported sensor measurement [m/s^2]

$\boldsymbol{f}^{\boldsymbol{b}}$ is the true acceleration [m/s^2]

$\boldsymbol{b}_{\boldsymbol{a}}$ is the bias vector [m/s^2]

$\boldsymbol{S}_{\boldsymbol{1}}$ is the scale factor error

$\boldsymbol{S}_{\boldsymbol{2}}$ is the scale factor non-linearity error

$\boldsymbol{N}_{\boldsymbol{a}}$ is the sensor axis non-orthogonality error

$\boldsymbol{\delta g}$ is the difference between the true and theoretical gravity value

$\boldsymbol{\varepsilon}_{\boldsymbol{a}}$ is the sensor noise [m/s^2]

It is evident that the sensor readings consist of the true motion undergone by the body modulated by a certain degree of error. That is, for the accelerometer,

$$\tilde{f}^b = f^b + e_a \tag{2}$$

where

$\tilde{\boldsymbol{f}}^{\boldsymbol{b}}$ is the reported sensor measurement [m/s^2]

$\boldsymbol{f}^{\boldsymbol{b}}$ is the true acceleration [m/s^2]

$\boldsymbol{e}_{\boldsymbol{a}}$ is the cumulative magnitude of all uncompensated accelerometer errors [m/s^2]

The readings from the accelerometer are integrated with respect to time to obtain velocity:

$$\tilde{v} = \int \tilde{f}^b \, dt = \int (f^b + e_a) \, dt = \int f^b \, dt + \int e_a \, dt = v + e_a t \tag{3}$$

where

$\tilde{\boldsymbol{v}}$ is the velocity computed from the accelerometer readings [m/s]

$\boldsymbol{v}$ is the true velocity [m/s]

Similarly, the velocity measurements are integrated with respect to time to obtain position:

$$\tilde{r} = \int \tilde{v}\, dt = \iint \tilde{f}^b\, dtdt = \iint (f^b + e_a)\, dtdt = \iint f^b\, dtdt + \iint e_a\, dtdt$$

$$= r + \tfrac{1}{2} e_a t^2$$

$$= r + \delta r \qquad (4)$$

where

$\tilde{\boldsymbol{r}}$ is the position computed from the accelerometer readings [m]

$\boldsymbol{r}$ is the true position [m]

$\boldsymbol{\delta r}$ is the position error [m]

From equations 3 and 4 we see that uncompensated errors in the accelerometer readings result in a velocity error that increases linearly with time and a corresponding position error that increases quadratically with time. Table 1 shows the position error with respect to time for various grades of accelerometers [4].

Table 1 – Exponential increase in horizontal position error with respect to time that arises from an uncompensated accelerometer bias.

Sensor Grade	Accelerometer Bias Error [mg]	Horizontal Position Error [m]			
		1 s	10 s	60 s	1 hr
Navigation	0.025	0.13 mm	12 mm	0.44 m	1.6 km
Tactical	0.3	1.5 mm	150 mm	5.3 m	19 km
Industrial	3	15 mm	1.5 m	53 m	190 km
Automotive	125	620 mm	60 m	2.2 km	7900 km

Following a similar analysis, the value reported by the gyroscope can be modeled as [1]:

$$\tilde{\omega}_{ib}^{b} = \omega_{ib}^{b} + b_g + S_g\omega_{ib}^{b} + N_g\omega_{ib}^{b} + \varepsilon_g \quad (5)$$

where

$\tilde{\boldsymbol{\omega}}_{\boldsymbol{ib}}^{\boldsymbol{b}}$ is the reported sensor measurement [deg/h]

$\boldsymbol{\omega}_{\boldsymbol{ib}}^{\boldsymbol{b}}$ is the true rotation rate [deg/h]

$\boldsymbol{b_g}$ is the bias vector [deg/h]

$\boldsymbol{S_g}$ is the scale factor error

$\boldsymbol{N_g}$ is the sensor axis non-orthogonality error

$\boldsymbol{\varepsilon_g}$ is the sensor noise [deg/h]

We can rewrite equation 5 as

$$\tilde{\omega}^{b} = \omega^{b} + e_g \quad (6)$$

where

$\tilde{\boldsymbol{\omega}}^{\boldsymbol{b}}$ is the reported sensor measurement [deg/h]

$\boldsymbol{\omega}^{\boldsymbol{b}}$ is the true rotation rate [deg/h]

$\boldsymbol{e_g}$ is the cumulative magnitude of all uncompensated gyroscope errors [deg/h]

The reported rotation rate $\tilde{\boldsymbol{\omega}}^{\boldsymbol{b}}$ is integrated with respect to time to obtain the total reported angle $\tilde{\boldsymbol{\theta}}$ through which the body has rotated [1]:

$$\tilde{\theta} = \int \tilde{\omega}^{b}\, dt = \int \left(\omega^{b} + e_g\right) dt = \int \omega^{b}\, dt + \int e_g\, dt$$

$$= \theta + e_g t$$

$$= \theta + \delta\theta \tag{7}$$

where

$\widetilde{\boldsymbol{\theta}}$ is the total rotation angle obtained using the reported sensor measurement [deg]

$\boldsymbol{\theta}$ is the true rotation angle [deg]

$\boldsymbol{\delta\theta}$ is the rotation angle error [deg]

As the obtained rotation angle $\widetilde{\boldsymbol{\theta}}$ is used to project the accelerations into the navigational reference frame, the error in the obtained rotation angle $\boldsymbol{\delta\theta}$ will induce an error component in the projected acceleration vector. The magnitude of this error for one axis is [2]:

$$\tilde{e}_a^p = gsin(\delta\theta) = g\delta\theta, \theta \ll 1 \tag{8}$$

where

$\tilde{\boldsymbol{e}}_{\boldsymbol{a}}^{\boldsymbol{p}}$ is the error in projected acceleration [m/s^2]

Integrating the error in projected acceleration $\tilde{\boldsymbol{e}}_{\boldsymbol{a}}^{\boldsymbol{p}}$ results in the error in projected velocity $\tilde{\boldsymbol{e}}_{\boldsymbol{v}}^{\boldsymbol{p}}$:

$$\tilde{e}_v^p = \int \tilde{e}_a^p \, dt = \int g\delta\theta \, dt = \int ge_g t \, dt = \frac{1}{2} ge_g t^2 \tag{9}$$

Integrating the error in projected velocity $\tilde{\boldsymbol{e}}_{\boldsymbol{v}}^{\boldsymbol{p}}$ results in the error in projected position $\tilde{\boldsymbol{e}}_{\boldsymbol{r}}^{\boldsymbol{p}}$:

$$\tilde{e}_r^p = \int \tilde{e}_v^p \, dt = \iint g\delta\theta \, dtdt = \iint ge_g t \, dtdt = \int \frac{1}{2} ge_g t^2 dt = \frac{1}{6} ge_g t^3 \tag{10}$$

From equations 9 and 10 we see that uncompensated errors in the gyroscope readings result in a velocity error that increases quadratically with time and a corresponding position error that increases cubically with time. It is this exponential error growth (or *drift*) that makes accurate long-

term inertial-only navigation a challenging problem that necessitates the use of additional sources of information to constrain and bound it.

The second, and much more significant, problem we face is due to the fact that a portable handheld navigation device is inherently untethered and free to be used in any three-dimensional orientation. Recall that the sensor misalignment error is the degree to which the sensor axes do not line up with the axes of the device body. If the body itself is not fixed to the platform in a specific orientation but is free to rotate as well (as is the case with a handheld navigation device such as a smartphone), the body axes cannot be guaranteed to line up with the platform axes. Therefore, we can define the *device misalignment error* as the degree to which the device axis does not line up with the axis of the platform. An incorrect device misalignment has a similar effect to an uncompensated gyroscope error as both lead to an incorrect projection of the accelerometer measurements. As such, an incorrect device misalignment leads to a velocity error that increases quadratically with time and a corresponding position error that increases cubically with time. Table 2 shows the position errors with respect to time that arises from an incorrect device misalignment [4].

Table 2 – Exponential increase in horizontal position error with respect to time that arises from an incorrect device misalignment.

Accelerometer Misalignment [deg]	Horizontal Position Error [m]			
	1 s	10 s	60 s	1 hr
0.05	4.3 mm	0.43 m	15 m	57 km
0.1	8.6 mm	0.86 m	31 m	110 km
0.5	43 mm	4.3 m	150 m	570 km
1	86 mm	8.6 m	310 m	1100 km

Long term accurate pedestrian navigation in GNSS-denied environments requires addressing the aforementioned issues. The unbounded error growth that arises from sensor drift can be mitigated through the use of additional sources of constraints and, more importantly, it is absolutely crucial that the device misalignment value can be reliably obtained. Much research has gone into using other sensors as a source of constraints and one particularly attractive sensor is the vision sensor, commonly referred to as the device camera. Vision/optical sensors offer a rich source of information about the environment and do not suffer from the effects of bias drift, making them particularly well suited for use in the realm of navigation.

2.2 Background to Pedestrian Dead Reckoning

There are many navigation scenarios in which the platform is a human moving on foot. In the case of pedestrian navigation, algorithm designers can take advantage of the periodic and repeatable nature of the human gait to simplify an otherwise complex INS mechanization. At its core, pedestrian navigation can be broken down into a series of steps or strides of a certain length in a specific direction [5]. After initialization from a known position and heading, knowing the stride length and platform direction allows one to update the user position after each detected stride. This process is known as *pedestrian dead reckoning*.

Mathematically, each detected stride updates the user position via:

$$x_k = x_{k-1} + s_L \sin(A_p) \tag{11}$$

$$y_k = y_{k-1} + s_L \cos(A_p) \tag{12}$$

where x_k and y_k are the x- and y-components of the updated position after step k, s_L is the detected stride length, and A_p is the azimuth or platform heading. In the global geodetic coordinate system, equations (11) and (12) become

$$\varphi_k = \varphi_{k-1} + \frac{s_L \sin(A_p)}{R_M + h} \tag{13}$$

$$\lambda_k = \lambda_{k-1} + \frac{s_L \cos(A_p)}{(R_M + h) * \cos(\varphi_k)} \tag{14}$$

where φ_k and λ_k are the geodetic latitude and longitude, respectively, R_M is the radius of the Earth's curvature, and h is the height above sea level.

In the case of PDR using a portable navigation device such as a smartphone, the device is free to change orientation with respect to the platform as shown in Fig. 2.2. Thus, the platform heading is given by

$$A_p = A_g - M \tag{15}$$

where A_p is the platform heading, A_g is the device heading, and M is the device heading misalignment, or the degree to which the platform and device heading are offset with one another.

The device heading can be obtained through integration of the angular rates reported from the gyroscope, leaving the correct value of the device misalignment a required parameter for obtaining an accurate platform heading. Though inertial-based methods for misalignment estimation exist, they are still subject to the inertial errors mentioned in the previous section and can fail with certain problematic gaits or at low walking speeds. As such, a non-inertial based method for misalignment estimation would be a valuable addition to a PDR-based navigation system. This makes vision-based aiding a particularly attractive means of obtaining device misalignment.

Though the position error in PDR-based schemes increases with the number of steps taken rather than with time as in an INS [5], it can be improved with additional information or constraints. A crucial component of any PDR system concerns accurate estimation of stride length. Stride length can be obtained using the step frequency [6] and magnitude of torso acceleration [7], among other means, but can change in different contexts. For example, if the device is being used

to send a text message, record video, or engage in any variety of non-meaningful motion (i.e. "fidgeting" in a navigation sense), the solution can be fixed to the previous position. As another intuitive example, the stride length used when walking or jogging on flat ground is generally greater than that used when traversing a series of stairs. In this situation, detecting that the user is on a set of stairs allows for a reduction in the estimated step length and keeps the solution from inaccurately projecting the user further ahead of where they actually are. Furthermore, a limitation of PDR systems is that they inherently provide a two-dimensional solution and require other means of obtaining user altitude. Following the stairs example, detecting that the user is undergoing motion on stairs allows for an estimation of changes in altitude. A vision-based aiding system would be a suitable means of classifying and supplying user context.

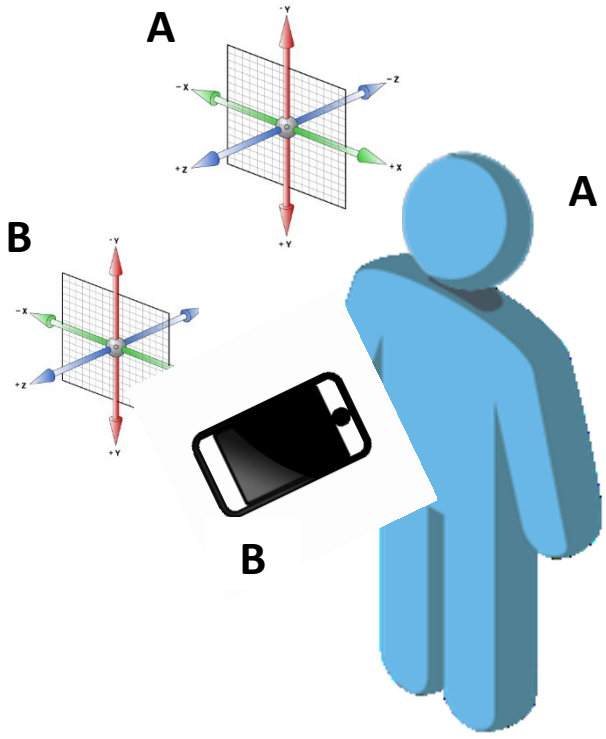


Fig. 2.2. Platform (A) and device (B) orientation. The platform heading can be derived using the device heading and device heading misalignment.

2.3 Previous Work in Vision-Based Navigation

Mautz and Tilch [8] conducted an extensive survey of vision-based indoor positioning systems and categorized both commercial and in-development systems based on the source of their reference systems for the surrounding environment. They classified the systems into one of six categories: (1) those using three-dimensional building floor plans as a reference; (2) those using a reference database of geo-tagged images; (3) those using coded targets placed in the environment as their reference; (4) those using targets projected into the environment as their reference; (5) systems without any reference source; and (6) systems that obtain their reference solution from other sensors such as an INS or GNSS receiver [8]. The following discussion highlights some of the systems described in their survey that are relevant to pedestrian navigation.

The first category of optical positioning systems is based on matching features found in the images with their pre-determined coordinates from a database [8]. These so-called map-based systems use an acquired set of landmarks that are expected to be found in a specific location to allow the navigation module to estimate its position [8]. To achieve real-time performance, these systems typically use a method of reducing the search space (e.g. using WiFi fingerprints) and can achieve accuracy on the decimeter-level [8]. The lowest-cost and portable example of such a system is found in the work of Hile and Borriello [9] who aim to match images taken from a smartphone with a known floor plan. The computational complexity and processing power available on a smartphone at the time of the research limited the frame rate to 0.1 Hz though dedicated hardware can be used to achieve rates up to 50 Hz with the associated increase in cost and reduction in portability [8]. Huang [10] used passive ranging to obtain the 3D positions of features in images and matched them with the 3D positions of the features in a known floor plan using the Random Sample Consensus (RANSAC) algorithm in an effort to derive the camera

position and orientation. The work showed an improvement in position accuracy as compared to WiFi positioning, reporting an accuracy of 5 meters on average, and was implemented on an Apple iPad tablet [10]. However, a major drawback of these types of systems is their failure in un-surveyed environments or when a floor plan is unavailable.

The opportunistic detection of environmental features within an image and matching to an existing database can falter in conditions of varying luminosity, obstruction/occlusion of these features, as well as in environments where no such database exists. The second category of systems attempt to overcome this problem by matching an entire image with a pre-surveyed series of images taken along an indoor pathway [8]. As these systems attempt to match entire images together rather than a set of detected features present in each, they are computationally-intensive and no real-time, portable, commercial deployment currently exists. The systems in development report sub-meter accuracy [8] and can be found in the work of Ido et al. [11]. Like the previous scheme, these systems also suffer from needing an *a priori* optical survey of the navigation environment with which images can be matched.

The third type of system relies on the detection of known reference markers placed in known locations in the environment [8]. An example would be the detection of a QR code sticker placed at a pre-surveyed location. A low-cost and portable example of such a system is developed by Mulloni [12] and can position a camera with sub-decimeter accuracy. However, deploying reference markers in the environment is a scheme that can be impractical. To overcome the infrastructure requirements associated with deployed markers, some researchers are looking at projecting markers into the environment using light [8]. Projected marker systems typically use a projected array of points and aim to find the central intersection of these points to obtain an estimate of the camera pose [8]. An example can be found in the laser-rig of Tilch and Mautz [13]

which achieves sub-millimeter accuracy and delivers an estimate of the camera pose at a frame rate of 30 Hz [8]. The need for additional hardware drives up the costs of projected systems and the accuracy of these systems range from the sub-millimeter to the sub-decimeter level [8].

The aforementioned systems are similar in that they derive the camera's position and orientation from markers or features that are at known locations (i.e. they are reference-based systems). However, cameras can be used as part of systems with no such reference by tracking the positional changes of the objects in the images directly [8]. An example of these so-called landmark systems is the DAEDALUS system developed by Burki et al. [14]. DAEDALUS provides sub-millimeter accuracy but is constrained to being mounted on a total station [8]. Such an expensive and bulky system cannot be used in a portable, handheld fashion. Furthermore, these systems aim to position the objects that are in the camera's field of view rather than the camera itself.

Lastly, vision-based systems can complement the positioning solution obtained by other systems to provide even greater accuracy. This is the category that this work falls under. Aufderheide and Krybus [15] use dedicated hardware to track features found between images and estimate the ego-motion parameters (position and orientation) of the camera. These parameters serve to bound the drift of the inertial-only solution while simultaneously reconstructing the surrounding scene. Ruotsalainein [16] computed and tracked the location of vanishing points throughout a sequence of images in order to obtain the heading change of a vision-aided pedestrian navigation system. This form of aiding is known as a visual gyroscope and was used to calibrate gyroscope errors [16]. The method involved the use of Gaussian smoothing to filter out high-frequency noise within the images, the Canny detector for edge detection, the Hough transform for straight line detection, as well as RANSAC for determination of the vanishing point itself [17].

With the exception of RANSAC, these same algorithms are used in this work albeit for context classification of the mobile device. Despite the heavy computational load, this system provided a 93% improvement to the heading error using a backpack set up and a 34% reduction in horizontal position error using an ankle set up [16].

Smartphones and personal navigation devices have only recently found widespread use and the body of work surrounding the role of computer vision for these specific devices is limited in comparison to the amount of research on vision systems for mobile robots. However, many of the same innovations used for robotic applications can be incorporated for use onto the smartphone platform. As such, this warrants a review of the pertinent literature. Guzel [18] published a survey similar to that of Mautz and Tilch with an emphasis on vision-based indoor navigation strategies for mobile robots. Bonin-Font et al. [19] published a more comprehensive survey of vision-based mobile robot navigation. The subsequent discussion follows the structure of and cites some examples from these two works. Bonin-Font et al. classified the strategies involved in robot navigation using computer vision into three major categories: (1) map based, (2) map-building based, and (3) mapless systems [19].

2.3.1 Map-based Systems

Map-based systems provide a moving platform such as a robot with a sequence of the environment features that are to be found and their corresponding locations on a map or database [18]. Computer vision serves to examine the incoming visual stream for known features and localizes the robot upon successful detection of these features [18]. This is the category that the first three types of optical navigation systems described by [18] fall into. Map-based systems are split into three sub-types: (a) absolute localization, (2) incremental localization, and (3) landmark

tracking [18]. Absolute localization assumes that the robot has no information regarding its initial position and requires either that the features in the incoming image stream match exactly with those found in the existing map/database [18] or the use of a Monte Carlo based method as outlined in [20]. Incremental localization assumes a coarse approximation of the robot's starting position and re-localizes when position errors exceed a set threshold [18]. Landmark tracking differs from the previous schemes in that, upon successfully identifying and detecting a landmark in the map/database, the landmark itself is tracked over subsequent images and used to derive the robot's position as opposed to re-querying the database [18].

2.3.2 Map-building Based Systems

Map-building based navigation aims to build a representation of the environment and is largely focused on structure from motion (SFM) methods using simultaneous localization and mapping (SLAM) [18]. The aim of SFM research is the accurate reconstruction of an assumed static three-dimensional scene from a series of two-dimensional images captured by a device moving through the environment [21]. Traditionally, SFM has been performed by detecting well-defined features in the images and tracking them throughout the video sequence. Each feature is uniquely identified and repeatedly localized within the scene. Because the scene is assumed to be static, positional changes of the features are assumed to be due to the movement of the capturing device and a solution for the device motion is obtained [21]. The disadvantage of SFM is the necessity for the offline, batch processing of all frames captured in the sequence. Furthermore, SFM fails in scenarios where the static scene assumption is violated [21].

SLAM can be performed with a variety of sensors such as IMUs or laser-range finders [21]. FootSLAM [22] relies on foot-mounted inertial sensors to estimate a user's location within a

building while the assumption of a user being unable to walk through walls allows for the simultaneous creation of the building floor plan. The authors observed an RMS position error on the order of a few meters in 10 minutes with near real-time performance. The authors propose that the improvements in MEMS technology will soon allow FootSLAM to be moved to a smartphone platform and that smartphones will constantly be updating their position and building a layout of the environment they are in.

While the idea of SLAM has been an ongoing area of research in the robotics community, there has been increasing focus into visual SLAM (i.e. SLAM using vision sensors) in recent years. Visual SLAM can be thought of as a real-time online SFM method that relies on the robust detection and matching of features within a sequence of images in order to simultaneously obtain the camera's position and orientation (or pose) and a map of the environment [23]. The traditional method of obtaining feature points and matching them between images prior to the estimation stage, be it a Kalman filter in EKF-SLAM [24] or a particle filter in FastSLAM [25], is known as feature-based visual SLAM [23] and usually employs the *scale invariant feature transform* (SIFT) [26] – a means of feature extraction that is robust against changes in image scale, rotation, brightness or camera view-point – or *speeded up robust features* (SURF) [27] – a faster and more robust improvement to SIFT – algorithms for feature detection. Feature-based methods are contrasted with direct methods to visual SLAM, where the image pixel intensities are used directly in the estimation stage to compute camera pose [23]. In a direct SLAM experiment using a synthetic scene, researchers were able to obtain a final drift of less than 0.001% of the total amount of translation and 0.091 degrees for rotation [23].

SLAM can be performed with a monocular or stereoscopic camera setup. Single camera SLAM, also known as MonoSLAM, with no motion constraints is a challenging problem. After

supplying their algorithm with an *a priori* starting position, the authors of [21] demonstrated successful real-time performance on a 2.2 Ghz Pentium processor in an environment with an average of 60 artificially-placed trackable features. They theorized that no more than 100 features could be tracked while maintaining real-time performance (i.e. a frame rate of 30 Hz). However, the authors noted that existing optimization schemes could increase this upper limit [28].

Wide angle cameras can also be used to improve performance. Davison [29] noticed improvements in the accuracy of their motion tracking algorithm by extending the camera field of view from 50 to 90 degrees. As these systems rely on tracking the motion of landmarks between frames, large and abrupt changes in camera position or orientation can cause landmarks to slip outside the field of view and lose observability. Wide angle cameras offer the benefit of reducing the perceived size of the motion so that larger accelerations and rotations can be handled [29]. However, wide lenses introduce distortion and camera calibration is a necessary step to preprocess the images before they are used [29].

2.3.3 Mapless Navigation

Mapless navigation uses features found in the environment using various techniques that do not involve using or building a map [18]. These systems can be further classified by the techniques they employ: optical flow based, appearance based, and qualitative based methods [18].

2.3.3.1 Optical Flow Based Methods

The first method – and foundational technique used in this work – is the calculation of optical flow, the apparent motion of brightness patterns within a sequence of images [18]. Optical flow has been used extensively for obstacle avoidance, terrain following, visual servo control, and

localization [30]. An example of obstacle avoidance using optical flow can be found in [31] whereby a camera equipped robot analyzes the flow pattern in each half of the images. Obstacles in the environment typically manifest themselves as regions of higher flow as compared to the image background [31]. As such, the robot balances the optical flow measured in each half of the image as a means of obstacle avoidance, opting to change its heading towards the side with a lower velocity (and hence, no obstacle) [18]. With the processing power available at the time of publication, the authors were able to obtain half real-time performance (12.5 Hz) on a 200 MHz Pentium processor [31]. Modern processors would be more than capable of accomplishing this task in real-time. The authors of [32] demonstrated a similar idea using both a monocular and stereo camera setup to successfully detect dynamic, rather than only static, obstacles. In the dual-camera setup, images from both cameras are used to construct the stereo disparity field and provide depth information while optical flow is calculated using images from one of the cameras [19]. Though experiments have shown successful results using this method, a problem with labelling areas of high optical flow obstacles is that irregularities in the image can be mislabeled as an obstacle and an otherwise correct navigation trajectory may be entirely avoided [19].

The authors of [30] use the average optical flow measured in the vertical as the input to a control system responsible for landing an unmanned aerial vehicle (UAV) with vertical take-off and landing (VTOL) capability on a moving platform with assumed bound dynamics. An onboard IMU was used for derotation of the images and the authors noted that the latency between inertial and visual information was a challenge in the implementation of their algorithm [30]. The authors were still able to successfully land the UAV in real-world experiments, though experiencing small unexpected oscillations in the height of the UAV on descent.

2.3.3.2 Appearance Based Methods

Appearance-based methods aim to create a working memory of the environment using either the full image or a representative template of the image using prominent landmarks [33]. These systems are split into two major types and require a priori information in some form. The first type, the model-based approach, uses pre-defined models of the features that are expected to be encountered in the environment [34]. The second type, the view-based approach, uses image matching algorithms that take the entire visible scene into account without the need for feature extraction [35]. The first phase of operation is known as the "learning phase", in which prominent environmental landmarks are localized and catalogued in a preliminary pass-through of the area [18]. In the context of robot navigation, these landmarks are associated with steering command as a means for the robot to traverse to the next set of landmarks from the currently observed set. After the learning stage, these systems operate in a navigation phase in which it compares the current observed image to the stored model of its environment and, for robot navigation, obtains the pertinent navigation information to proceed to the next visual "checkpoint" [18].

An example of a model-based appearance-based system can be found in the work of Courbon et al. [36] in which key images of the environment are taken by a UAV and organized to provide a visual memory of the traversed path. The navigation module then links the current observed image to the target image in the database by searching for landmarks in the former and matching them with those found in the latter. Sub-meter positional accuracy has been obtained using such a scheme [36]. A variation of the aforementioned scheme uses an omnidirectional camera pointed upwards at a mirror to obtain an orthographic view of the ground surrounding a robot [37]. The advantage of using such a camera is the elimination of any image distortion that would arise from

the effects of perspective however a sacrifice in resolution must be made for a wider field of view [38]. Again, this system reported sub-meter accuracy [37].

The argument for a view-based approach is based on the premise that recording views and using existing image-matching algorithms is a less challenging task than that of modeling the objects found in the environment and searching for them in noisy images [35]. The traditional disadvantage to this approach lay in the large memory and processing cost involved but the development of computing technology is making this an increasingly feasible approach [35]. The authors of [35] use a block matching process whereby the image in memory is the template and the currently observed image is the search space. The goal is minimization of the matching error between the two images [35]. As expected, the matching error increases and resets in between scenes that line up perfectly [35]. This behavior is similar to an INS system using external updates to reset the accumulated position errors from sensor drift. Experiments were able to show adequate matching with bounded errors for trajectories up to 12 meters in length [35].

2.3.3.3 Qualitative Based Methods

Appearance-based methods for obstacle avoidance require some degree of initial information as to the set of features that correspond to an obstacle. Sensor-based obstacle avoidance systems aim to use raw sensor information to distinguish obstacles from free space while avoiding the explicit computation of numerical navigation information such as exact position and velocity or exact distances to obstacles [19]. This is referred to as qualitative navigation [19]. As complex calculations are avoided, this necessitates the need for a module that makes decisions based on the qualitative parameters obtained from the images. An example of this type of system can be found in the work of [39] whereby an obstacle avoidance system was designed based on the examination

of changes in illumination, RGB (red, green, blue) color, and HSV (hue, saturation, value) from the incoming image stream [19]. The results were fed into a decision-making module that generated motion commands to keep the robot moving in areas of free space [19]. An advantage to these types of systems is the avoidance of particularly complex computations (e.g. computing optical flow) as well as the low resolution images that can be used since the end goal is not to obtain an exact number representing a certain facet of the environment. However, the use of raw qualitative data can hamper performance in less-than-ideal conditions (i.e. unexpected changes in illumination) [19]. As most navigation systems target a certain degree of precision, accuracy, and robustness, qualitative systems can be used as coarse complement or starting point for a higher-end system rather than the sole means of navigation.

An example of the qualitative nature of these systems can be found in [40] and is termed *fuzzy navigation*. In this work, edge detection and terrain slope calculation algorithms were applied to a sequence of images captured by a stereoscopic camera setup to classify objects encountered in an outdoor environment and react accordingly [40]. As an example, a horizon-line extraction algorithm was used to identify the boundary between the ground plane and the sky [40]. An edge-detection and region-growing method was applied to identify obstacles such as rocks that lay on the ground plane [40]. The presence or absence of these obstacles gave an indication of terrain roughness, or the coarseness and irregularity of the ground surface [40]. After classifying whether the terrain was “smooth”, “rough”, or “rocky”, the obstacles found on the ground plane were classified as “large” or “small” according to their size, by retrieving the pixel count within the boundaries of the identified object and comparing to a threshold value [40]. The average separation between the objects were used to classify the obstacles as “few” or “many” while the depth information provided by the stereo setup gave an indication of whether the obstacles were “near”

or “far” and whether the terrain was “flat”, “sloped”, or “steep” [40]. The combination of these labels allowed for the creation of a rule-base with matrix that gave an indication of the terrain’s traversability [40]. For example, regions deemed “flat” and “smooth” result in a “high” traversability while those deemed “steep” and “rocky” result in a “low” traversability [40]. These qualitative classifications were then used to generate motion commands that would guide the robot into regions of greater traversability [40].

The following chapters will demonstrate how some of the aforementioned techniques (namely optical flow) can be used to provide vision-aiding for a PDR-based system in indoor environments.

Chapter 3: Methodology of Device Angle Calculation Module

The stages of the vision-aiding algorithm are as outlined in Fig. 3.1 [41]. The device angle calculation sub-module is outlined in the upper rectangle and explained in this chapter [41]. The context classification sub-module is in the lower rectangle and will be addressed in Chapter 4.

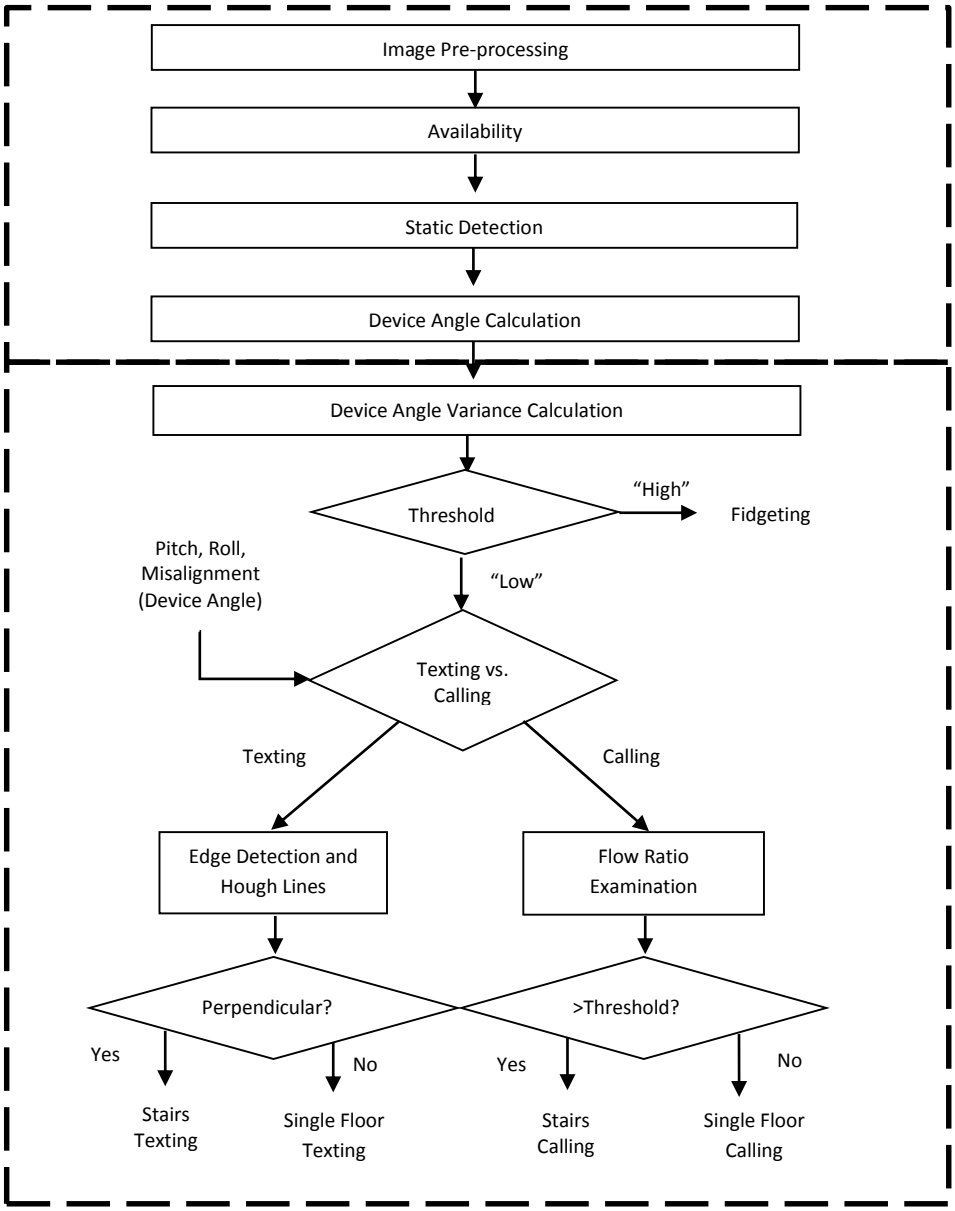


Fig. 3.1. High level overview of the vision-aiding module.

3.1 Image Pre-processing

Image processing is a computationally intensive task and optimizations must be made if the derived algorithms are intended to be used in real-time. Furthermore, the amount of light entering a vision sensor is highly variable and dependent on an environment that the designer has little to no control over. These facts necessitate the need to pre-process the images being input into the device angle calculation module. The pre-processing stage consists of four sub-stages: (1) size reduction, (2) grayscale conversion, (3) histogram equalization, and (4) image smoothing.

3.1.1 Size Reduction

Modern digital cameras are capable of acquiring images at an unprecedentedly high size and resolution. These images are typically at a minimum size of 1280x720 pixels and 24-bit color depth (i.e. bits per pixel). At a size of 1280x720 pixels, the optical flow routine is the primary bottleneck in the algorithm runtime, taking nearly two seconds to process each frame on a PC equipped with a 2.3 Ghz Core i3 processor and 4 megabytes of RAM. Real-time video is typically captured at 30 frames per second (fps). The results of runtime tests performed on the PC using images of various sizes are shown in Fig. 3.2. Examining Fig 3.2, it is apparent that only at 10% of the original 1280x720 image resolution can the optical flow routine run at 30 fps on the aforementioned PC. This reduction in image size necessarily involves some loss of information within the image so care must be taken that the images are reduced to a size suitable for real-time performance while the calculations still maintain a strong correspondence with the values obtained using the original full-size images. Fig 3.3 shows the value of the device angle calculated using optical flow measurements with images of various sizes. It can be seen that there is still a great degree of correspondence for the obtained device angle across all reduced image sizes. The

information provided by Fig. 3.2 and 3.3 shows that by reducing the image size to 128x72 pixels (i.e. 10% of the original resolution) we obtain a 60 times speedup in the calculation with no significant loss of information.

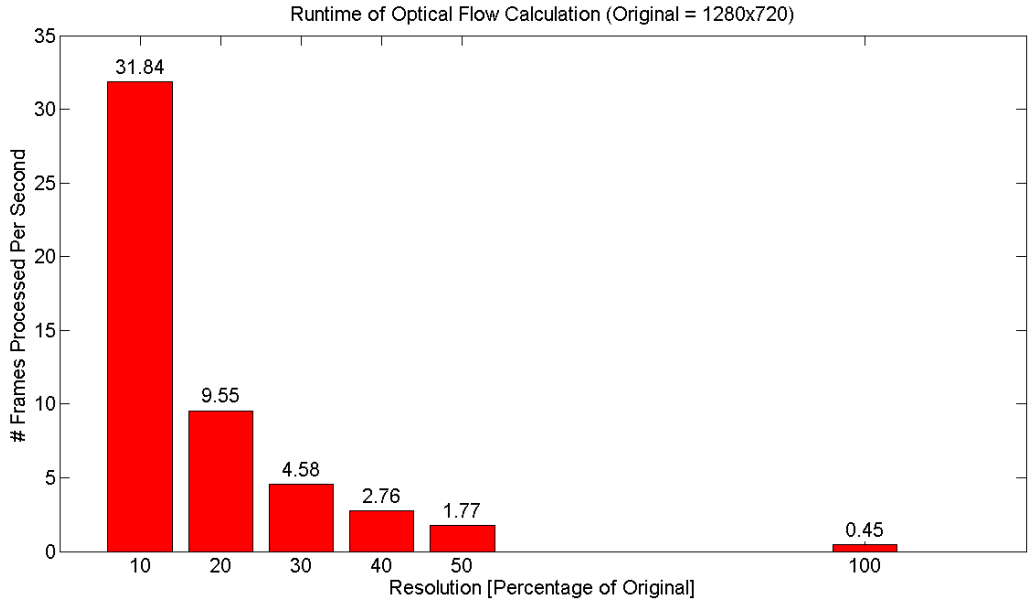


Fig 3.2. Runtime of the optical flow routine using various image sizes.

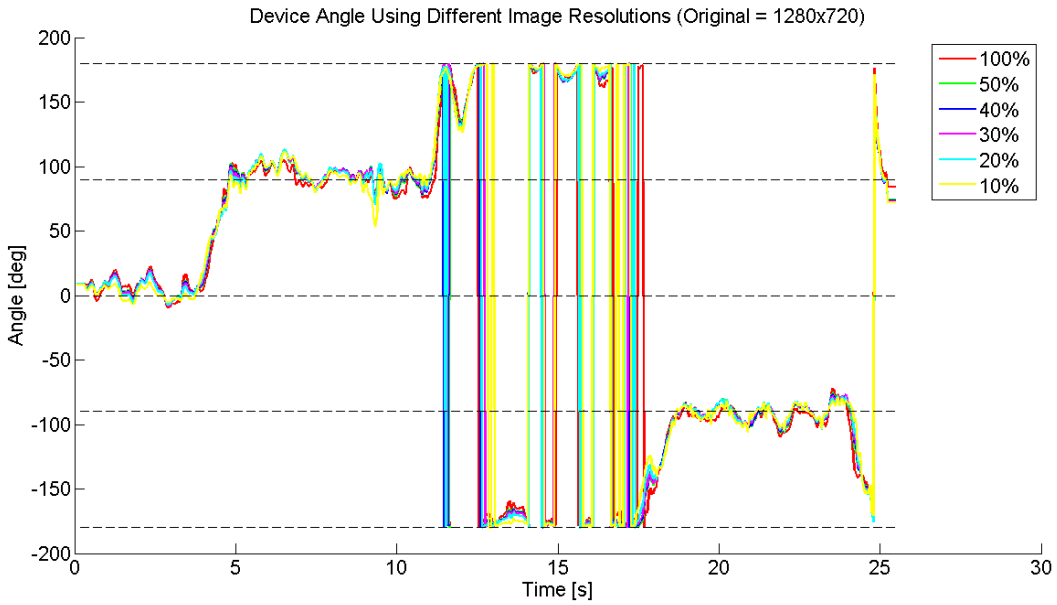


Fig 3.3. Value of device angle obtained using various image sizes.

3.1.2 Grayscale Conversion

Grayscale conversion is a necessary pre-processing step as it serves to radically reduce the complexity of the optical flow calculation. Rather than computing optical flow for each of the red, green, and blue colour channels that comprise a digital image, the information from all three channels can be represented as a single grayscale image. Grayscale conversion reduces the size of the matrices used in many areas of the entire algorithm by decomposing a 24-bit colour image into an 8-bit grayscale image. The conversion proceeds as follows.

Let $I_k(x,y)$ and $I_{k-1}(x,y)$ represent two color images of size N_x by N_y within an image sequence where x and y represent the spatial coordinates within the image and k represents the temporal coordinate of the frame within the sequence [42]. The value at every (x,y) location in I_k and I_{k-1} is a triplet whose individual elements indicate the 8-bit integer (i.e. 0-255) intensity value of each of the red, green, and blue (RGB) color channels in the form (r,g,b). In order to simplify the next step, we can equivalently represent each image by three N_xxN_yx1 arrays. Each array holds the 8-bit intensity value for one of the three RGB color channels. That is, the three-channel colour image $I_k(x,y)$ can be decomposed into the single-channel (i.e. single-colour) red, green, and blue arrays $r_k(x,y)$, $g_k(x,y)$, and $b_k(x,y)$, respectively.

Let G_k and G_{k-1} represent single-channel grayscale images that were obtained from the split-channel color images as per the formula used in the NTSC and PAL television standards [43]:

$$G_k(x,y) = 0.299r_k(x,y) + 0.587g_k(x,y) + 0.114b_k(x,y) \qquad (11)$$

where $i = 0, 1, \ldots, N_x\text{-}1$ and $j = 0, 1, \ldots N_y\text{-}1$.

3.1.3 Histogram Equalization

After conversion to grayscale, histogram equalization is applied to increase the image contrast. As optical flow aims to measure the translation undergone by regions of brightness in the image, having a greater distinction between these regions serves to improve the flow calculation. The process begins with the creation of the image histogram–a count of the number of pixels at each of the 256 intensity values. The equalization routine then creates a new image with a "stretched" version of the histogram so as to span the full intensity range (0-255) rather than being clustered tightly around a somewhat central value. More intuitively, this serves to convert a predominantly grey image to one that spans the entire grayscale palette from black to white.

The mathematical description of the histogram equalization process follows the description in [44]. Let *f* be an N_x by N_y matrix (representing an image) of integer pixel intensities ranging from 0 to $I_{max} - 1$ where I_{max} is the number of possible intensity values [44]. In a single-channel 8-bit image, I_{max} has a value of 256. Let p denote the normalized histogram of *f* with a bin for each possible intensity so that [44]:

$$p_n = \frac{number\ of\ pixels\ of\ intensity\ n}{total\ number\ of\ pixels} \qquad n = 0, 1, \ldots, I_{max} - 1 \tag{12}$$

The histogram equalized image *g* will be defined by [44]:

$$g(x, y) = floor((I_{max} - 1) \sum_{n=0}^{f(x,y)} p_n) \tag{13}$$

where *floor()* rounds down to the nearest integer. This is equivalent to transforming the pixel intensities, *k*, of *f* by the function [44]:

$$T(k) = floor((I_{max} - 1) \sum_{n=0}^{k} p_n) \tag{14}$$

An example of an image histogram before and after equalization using the MATLAB 'histeq' function on a sample image is given in Fig 3.4.

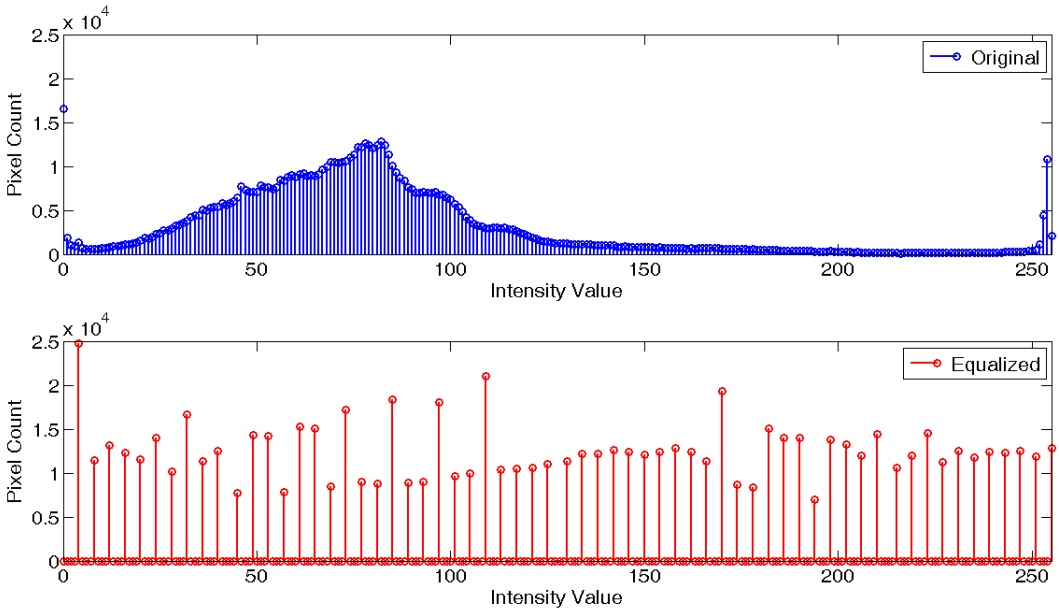


Fig. 3.4 – Histogram of an image before and after equalization. The histogram of the original image shows that the pixel intensities are distributed narrowly around a value of 80. After equalization, the values span the full range of 0-255.

3.1.4 Image Smoothing

An unavoidable consequence of increasing contrast via histogram equalization is that noise is enhanced as well, effectively lowering the signal to noise ratio in the image. This noise is generally of a high spatial frequency and can be removed by lowpass filtering or smoothing the image.

Image smoothing is performed using two-dimensional convolution. Convolution is an operation on two functions *f* and *g*, which produces a third function that can be interpreted as a modified ("filtered") version of *f*. In the case of image smoothing, we call *f* the image and *g* the "smoothing

kernel". The result of the convolution is a smoothed copy of the image *f*. Formally, for functions *f* and *g* of a continuous variable *x*, convolution is defined as:

$$f(x) * g(x) = \int_{-\infty}^{\infty} f(\tau)g(x-\tau)d\tau \tag{20}$$

where * means convolution. For functions of a discrete variable *x*:

$$f[x] * g[x] = \sum_{k=-\infty}^{\infty} f[k]g[x-k] \tag{21}$$

For functions of two variables *x* and *y* (e.g. images), these definitions become:

$$f(x) * g(x) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\tau_1,\tau_2)\, d\tau_1 g(x-\tau_1, y-\tau_2)d\tau_2 \tag{22}$$

and

$$f[x,y] * g[x,y] = \sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty} f[k_1,k_2]g[x-k_1, y-k_2] \tag{23}$$

Many types of smoothing schemes exist (homogenous, Gaussian, median, bilateral, etc.) and each is suited for noise of a particular distribution. In the case of homogenous smoothing, averaging the two neighbouring pixels in each direction would be performed using the 5x5 convolution kernel *K*:

$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

In this work, it was assumed that the image noise exhibited a Gaussian distribution and so Gaussian smoothing was applied. The appropriate kernel to be used was found using the 2D Gaussian function *G:*

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{24}$$

where

x and y are the distances from the origin in the horizontal and vertical axes, respectively

σ is the standard deviation of the Guassian distribution

As an example, using the Gaussian function given in equation (24) to build a 5x5 smoothing kernel setting $\sigma = 1$, we obtain the following kernel with which to convolve with the image:

$$K = \frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

The smoothed image is given by the convolution of the grayscale image with the smoothing kernel. The image is now ready to be analyzed and Fig 3.5 shows the effect of each preprocessing stage.

Fig 3.5. Effects of the various preprocessing stages. Original image, after grayscale conversion, after histogram equalization, and after smoothing (clockwise from top).

3.2 Availability and Static Determination

Availability is a measure of the homogeneity of an image (i.e. a measure of how feature-rich an image is) and is based on the derivative, or gradient, of the image. The reason for checking availability in the first stage of the algorithm is to ensure that the captured images have sufficient features to be used for reliable optical flow measurements. The first spatial derivative of an image can be obtained by convolving the image with a filter such as the Sobel filter and aggregating the result across the image [45]. In the case of the Sobel filter, the operator uses two 3×3 kernels which

are convolved with the original image to calculate approximations of the horizontal and vertical derivatives. If we define **f** as the source image, the horizontal and vertical approximations $\mathbf{G}_x$ and $\mathbf{G}_y$ are given as [45]:

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \mathbf{f} \tag{25}$$

$$\mathbf{G}_y = \begin{bmatrix} -1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & 2 & 1 \end{bmatrix} * \mathbf{f} \tag{26}$$

where * here denotes the 2-dimensional convolution operation described in the image smoothing section above.

The resulting gradient approximations in each direction can be combined to give the overall gradient magnitude **G**:

$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2} \tag{27}$$

Using this information, we can calculate the aggregated gradient *D* across the entire image:

$$D = \sum_{j=0}^{N_y-1} \sum_{i=0}^{N_x-1} \mathbf{G}[i,j] \tag{28}$$

The aggregated derivative value *D* can be normalized by dividing by the maximum intensity change between pixels (i.e. 0-255) in the entire image to get D_{norm}:

$$D_{norm} = \frac{D}{(N_x * N_y) * 255} \tag{29}$$

A small normalized aggregated derivative value, D_{norm}, implies high homogeneity and a lack of features that can be used for the subsequent optical flow calculations. A high derivative value implies a less homogeneous image (i.e. greater variation with the image). Large intensity

fluctuations are indicative of objects being within the field of view. The presence of multiple objects allows for a more reliable optical flow estimate.

Static detection is performed based on the value of the Mean Structural Similarity Index (MSSIM). The MSSIM is a singular value that quantifies the degree to which two images match in luminance, contrast, and structure [46]. For two input signals **x** and **y**, the SSIM within a window *m* of size *N*x*N* is given as [46]:

$$SSIM(x_m, y_m) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{30}$$

where

μ_x and μ_y are the means of x_m and y_m

σ_x^2 and σ_y^2 are the variance of x_m and y_m

σ_{xy} is the covariance between x_m and y_m

C_1 and C_2 are stabilizing coefficients

Taking the mean value across all *M* windows used to construct the structural similarity array, the MSSIM is given as [46]:

$$MSSIM(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sum_{m=1}^{M} SSIM(x_m, y_m)}{M} \tag{31}$$

An MSSIM value of zero indicates two completely unrelated pictures while a value of one indicates that the pictures are identical. Because a static feature-rich scene and a moving feature-less scene would both result in a high MSSIM, it is not sufficient to check the MSSIM value alone for static detection. If the MSSIM value of the images are found to be above a set threshold, the algorithm proceeds with a check on the homogeneity. High homogeneity (i.e. small derivative value) scenes indicate that the images will not provide meaningful optical flow information as they

are largely uniform. High derivative values indicate that the scene is feature-rich and can be used for a reliable optical flow estimate.

3.3 Optical Flow Calculation

The smoothed grayscale images are supplied as arguments to the optical flow calculation routine. The output of the optical flow routine is based on the brightness constancy constraint, which is the assumption that changes in pixel intensities are only due to small translational motions in the time interval between images [47]. Formally, brightness constancy is the assumption that a pixel at location (x, y, t) with intensity $I(x, y, t)$ will have moved by Δx and Δy in the time interval between images Δt [47]:

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t) \tag{32}$$

Assuming the movement to be small, the brightness constancy equation can be expanded using a Taylor series. To a first-order approximation:

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t \tag{33}$$

It follows that:

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0 \tag{34}$$

Dividing each term by the time interval Δt:

$$\frac{\partial I}{\partial x}\frac{\Delta x}{\Delta t} + \frac{\partial I}{\partial y}\frac{\Delta y}{\Delta t} + \frac{\partial I}{\partial t}\frac{\Delta t}{\Delta t} = 0 \tag{35}$$

yields

$$\frac{\partial I}{\partial x}u_x + \frac{\partial I}{\partial y}u_y + \frac{\partial I}{\partial t} = 0 \tag{36}$$

where u_x and u_y are the x and y components of the velocity and referred to as the optical flow, $\frac{\partial I}{\partial x}$ and $\frac{\partial I}{\partial y}$ are the spatial derivatives of the image intensities and $\frac{\partial I}{\partial t}$ is the temporal derivative.

This is an equation in two unknowns and cannot be solved without another set of equations. This is otherwise known as the "aperture problem". One widely used approach to solving the aperture problem, and the one used in this work, is the tensor-based Farneback method using polynomial expansion as described in [48]. Farneback's method approximates the neighbourhood surrounding each pixel in the frame using second-degree polynomials and estimates displacements from the way in which these polynomials change under translation [48].

Polynomial expansion aims to approximate the area surrounding each pixel with a quadratic polynomial f_1 of the form [48]:

$$f_1(\mathbf{x}) = \mathbf{x}^T\mathbf{A}_1\mathbf{x} + \mathbf{b}_1^T\mathbf{x} + c_1 \tag{37}$$

where **A** is a symmetric matrix, **b** a vector and c a scalar. The three polynomial coefficients **A**, **b**, and c are estimated from a weighted least squares fit to the intensity values in **x** (the area of the image that is being approximated) [48]. The weighting has two components called certainty and applicability that determine which regions of the image are involved in the calculation and the size of the structures that will be represented by the expansion coefficients, respectively [48]. Letting f_1 undergo an ideal global displacement **d**, we obtain f_2 [48]:

$$\begin{aligned} f_2(\mathbf{x}) = f_1(\mathbf{x}-\mathbf{d}) &= (\mathbf{x}-\mathbf{d})^T\mathbf{A}_1(\mathbf{x}-\mathbf{d}) + \mathbf{b}_1^T(\mathbf{x}-\mathbf{d}) + c_1 \\ &= \mathbf{x}^T\mathbf{A}_1\mathbf{x} + (\mathbf{b}_1 - 2\mathbf{A}_1\mathbf{d})^T\mathbf{x} + \mathbf{d}^T\mathbf{A}_1\mathbf{d} - \mathbf{b}_1^T\mathbf{d} + c_1 \\ &= \mathbf{x}^T\mathbf{A}_2\mathbf{x} + \mathbf{b}_2^T\mathbf{x} + c_1 \end{aligned} \tag{38}$$

Equating the coefficients of f_1 and f_2 [48]:

$$\mathbf{A}_2 = \mathbf{A}_1 \tag{39}$$

$$\mathbf{b}_2 = \mathbf{b}_1 - \mathbf{2A}_1\mathbf{d} \tag{40}$$

$$c_2 = \mathbf{d}^T\mathbf{A}_1\mathbf{d} - \mathbf{b}_1^T\mathbf{d} + c_1 \tag{41}$$

Re-arranging equation (41), we can solve for the **t**ranslation $\mathbf{d}$ [48]:

$$\mathbf{d} = -\frac{1}{2}\mathbf{A}_1^{-1}(\mathbf{b}_2 - \mathbf{b}_1) \tag{42}$$

3.4 Aggregate Image Translation

The output of the optical flow routine is the flow map $F_k(x,y)$ where each element is a tuple indicating an estimate of the translational motion (dx,dy) undergone by the intensity value at location (x,y) between images G_k and G_{k-1}. An exemplary flow map is given in Fig. 3.6. The dual-channel flow map F can be split into two single-channel matrices $dx_k(x,y)$ and $dy_k(x,y)$. All x- and y-components of the flow map elements are summed to provide the aggregate translation, $dx_{agg,k}$ and $dy_{agg,k}$, between images G_k and G_{k-1} respectively. That is,

$$dx_{agg,k} = \sum_{j=0}^{N_y-1}\sum_{i=0}^{N_x-1} dx_k[i,j] \tag{43}$$

$$dy_{agg,k} = \sum_{j=0}^{N_y-1}\sum_{i=0}^{N_x-1} dy_k[i,j] \tag{44}$$

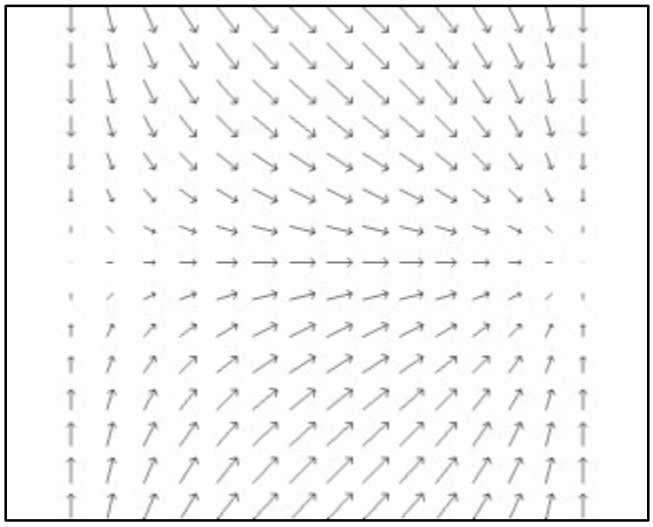


Fig. 3.6. Exemplary flow map.

3.5 Use Case Classification

The goal of use case classification is to distinguish between periodic and non-periodic motion. To this end, the values of the aggregate translation are used to create the binary thresholded signals

$$dx_{agg,k}^{binary} = \begin{cases} 1\ if\ dx_{agg,k} > 0 \\ -1\ if\ dx_{agg,k} < 0 \\ dx_{agg,k-1}^{binary}\ if\ dx_{agg,k} = 0 \end{cases} \tag{45}$$

$$dy_{agg,k}^{binary} = \begin{cases} 1\ if\ dy_{agg,k} > 0 \\ -1\ if\ dy_{agg,k} < 0 \\ dy_{agg,k-1}^{binary}\ if\ dy_{agg,k} = 0 \end{cases} \tag{46}$$

Differencing the binary thresholded flow from the previous step allows us to determine the transition points (if any) between forward and backwards movement:

$$D_{x,k} = dx_{agg,k}^{binary} - dx_{agg,k-1}^{binary} \tag{47}$$

$$D_{y,k} = dy_{agg,k}^{binary} - dy_{agg,k-1}^{binary} \tag{48}$$

Periodic spikes should be observed within the differenced signal from the previous step if the device is dangling. In other words, we can determine the dangling use case by looking for peaks in the magnitude spectrum *M* of the Fourier transform *Y* of the differenced signal *D* over a window of *N* images where:

$$Y_{x,m} = \sum_{j=0}^{N-1} D_{x,j}\omega_N^{(j-1)(m-1)} \tag{49}$$

$$Y_{y,m} = \sum_{j=0}^{N-1} D_{y,j}\omega_N^{(j-1)(m-1)} \tag{50}$$

where $\omega_N = e^{(-2\pi i)/N}$ is an *N*th root of unity, and the magnitude spectrum is given by

$$M_x = 2\left|Y_{x,m}\right| \tag{51}$$

$$M_y = 2\left|Y_{y,m}\right| \tag{52}$$

If a peak is found in the magnitude spectrum above a set threshold in the dangling frequency range (experimentally determined to be between 0.5 and 1.5 Hz for slow to fast walking speeds due to the natural swinging of the human arm), the use case is classified as dangling. The frequency of the peak is taken as the dangling frequency, $f_{dangling}$. If a peak was not found, the use case is classified as texting or vertical.

The texting use case is distinguished from the vertical use case by examining the pitch and roll of the device. A device held in the portrait orientation is classified as texting when exhibiting a pitch below 45 degrees and vertical when greater than 45 degrees. In the landscape orientation, the device roll is used in place of pitch.

3.6 Device Angle Calculation

Once the use case has been determined, device angle calculation can proceed. The method for calculating the device angle differs based on the determined use case of the device and is only deemed a reliable value for heading misalignment when exhibiting a low variance.

3.6.1 Texting Use Case

If the use case has been classified as texting, the device angle, θ_k, is computed using the values of the aggregate x- and y-translations:

$$\theta_k = \text{atan2}(dy_{agg,k}, dx_{agg,k}) \tag{53}$$

An example of common texting use case misalignments can be found in Fig 3.7.

Fig 3.7. Device angles of 0, 90, 180, and -90 degrees with respect to the direction of motion indicated by the black arrow (shown left to right).

3.6.2 Vertical Use Case

In the vertical use case (e.g. walking down a hallway), using the entire image for the optical flow calculation will lead to incorrect values for the device angle. In the texting and calling use

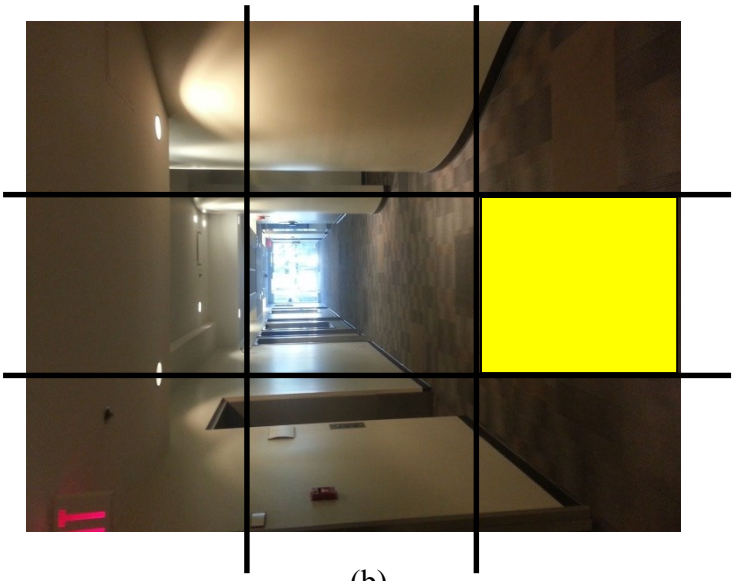
cases, all pixels in the image undergo a translation in approximately the same direction and this allows the entire image to be used for the calculation of device orientation. However, in the vertical use case, the optical flow routine returns a non-uniformly directed flow map that diverges outwards from a central location known as the "vanishing point" of the image. An example is found in Fig. 3.8.

As it is the translation undergone by the floor region that is indicative of the device orientation, the area of the image representing the floor must be identified and used in the optical flow calculation to obtain the correct value of the device angle. A simple and effective strategy for determining the location of the floor region is to partition the image into a 3x3 grid of equally-sized cells. With the device held at an angle of zero degrees, the floor is typically in the bottom-most middle cell. Fig. 3.9 shows the correct partitions for two common orientations.

$$\theta_k = \text{atan2}(dy_{agg,k}^{floor}, dx_{agg,k}^{floor}) \tag{54}$$

Fig 3.8. Uniform optical flow field obtained when the device is in the texting use case, and divergent optical flow obtained when the device is in the vertical use case.

(a)



(b)

Fig 3.9. Camera view when the device is held in the vertical use case at an angle of (a) zero degrees and (b) 90 degrees. The appropriate cell to use for the optical flow calculation is highlighted.

3.6.3 Dangling Use Case

If the use case has been classified as dangling, the device angle calculation proceeds as follows. The differencing of the binary thresholded aggregate flow (D_x and D_y) and/or the maximum and minimum peaks in either the pitch or roll signals from the device's on-board IMU are used to obtain the transition points between the start of the forward and backward dangling half-cycles.

If the magnitude of the cumulative x flow over the full cycle is *greater* than that of the cumulative y flow over the full cycle and/or if the *pitch* is found to have a larger periodic signal than roll and/or if the x-flow exhibits greater periodicity than the y-flow as shown by the magnitude spectrum of the differenced binary thresholded signal), the absolute values of the aggregated x-flows in each half-cycle are integrated. If the cumulative x-flow, dx_{cum}, during the positive half cycle (PHC) is greater than the cumulative x-flow in the negative half-cycle (NHC), this indicates that the device is oriented in the direction of the motion and the device angle is taken to be 0 degrees. Otherwise, the device is oriented against the direction of motion and the device angle is taken to be 180 degrees. That is,

$$\theta_k = \begin{cases} 0 \; if \; dx_{cum} in \; PHC > dx_{cum} in \; NHC \\ 180 \; else \end{cases} \tag{55}$$

If the magnitude of the cumulative x flow over the full cycle is *less* than that of the cumulative y flow over the full cycle and/or if the *roll* is found to have a larger periodic signal than pitch and/or if the y-flow exhibits greater periodicity than the x-flow (as shown by the magnitude spectrum of the differenced binary thresholded signal), the absolute values of the aggregated y-flows in each half-cycle are integrated. If the cumulative y-flow during the PHC is greater than the cumulative y-flow in the NHC, the device angle is taken to be -90 degrees. Otherwise, the device angle is taken to be +90 degrees. That is,

$$\theta_k = \begin{cases} -90 \; if \; dy_{cum} in \; PHC > dy_{cum} in \; NHC \\ 90 \; else \end{cases} \tag{56}$$

A state machine can be used to keep track of whether we are currently in the positive or negative half cycles of the dangling motion. An example state machine can be found in Fig. 3.10. Beginning in the initialization state, a positive spike in the differenced binary thresholded flow signal (a value of +2) indicates that the device is experiencing a change from negative to positive flow and transitions the system to the PHC state. During the PHC state, a negative spike in the differenced binary thresholded flow signal (a value of -2) transitions the system to the NHC state, whereupon a positive spike transitions the system back to the PHC state. The x- and y-flow values are integrated in each half-cycle and used in accordance with equations (50) and (51) to obtain a value for the dangling use case device angle.

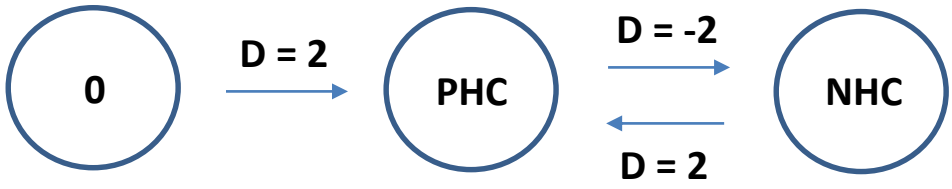


Fig. 3.10. Example state machine used for determining positive and negative half-cycles of the dangling motion.

Chapter 4: Methodology for Context Classification Module

After the images have been processed through the device angle calculation sub module, they proceed to the context classification sub module described in this chapter. This sub module aims to discern meaningful device motion from fidgeting as well as device motion on a single floor versus scenarios in which the device is undergoing a significant height change (e.g. a floor change) in the texting or calling use cases. The variance of the vision-derived device misalignment value is the basis for detecting the "fidgeting" context while the ratio of the components of the optical flow and presence of lines in the image are used to discern motion on a single floor from motion on stairs. The details of context classification are as follows [49].

4.1 Fidgeting

It is important to classify whether the motion undergone by the device is meaningful in a navigational sense or if the device is undergoing a variety of navigationally irrelevant motion (e.g. playing games, sending a message, interacting with a map on the device, etc.). The variance of the device angle obtained using the values of the aggregated optical flow is used to distinguish meaningful motion (relatively low device angle variance) from fidgeting (relatively high device angle variance). It is important to note that it would be incorrect to use the standard (i.e. arithmetic) variance formula as the device angle is inherently a circular quantity. A simple example highlighting this fact can be demonstrated by taking the arithmetic (as opposed to circular) mean of the angles 0 and 360 degrees (identical angles). The standard (i.e. arithmetic) mean formula incorrectly gives a value of 180 degrees whereas the circular mean formula gives a value of zero degrees. By the same token, we must examine the normalized circular variance of the device angle, V, over a window of N samples, as per [50]:

$$R^2 = \left(\sum_{i=1}^{N} \cos\theta_i\right)^2 + \left(\sum_{i=1}^{N} \sin\theta_i\right)^2 \tag{57}$$

$$V = 1 - \frac{R}{N} \tag{58}$$

Comparing the variance or standard deviation of the device angle to a threshold value, we can determine if the phone is undergoing fidgeting or meaningful motion. Again, it is important to note that because of the circular nature of the data, it would be incorrect to take the square root of the variance as the standard deviation [50]. The circular standard deviation, *v*, is given by [50]:

$$v = \sqrt{-2\ln(1 - \frac{R}{N})} \tag{59}$$

4.2 Texting vs. Calling Use Case Classification

Pitch, roll, and misalignment are used to distinguish the texting use case from the calling use case. In the first case, if the pitch is near 90 degrees and misalignment is around 90 or -90 degrees, the device is speaker up with the camera to the side. In the second case, if the roll is near 90 or -90 degrees and the misalignment is around 0 or 180 degrees, the device is held with the speaker backwards or forwards (respectively) and the camera is to the side. In both cases the device is in the calling use case. If neither case is satisfied, the device is in the texting use case.

4.3 Texting Use Case – Single Floor vs. Stairs

The single floor texting use case can be distinguished from the stairs texting use case by performing edge detection and searching for spatially periodic groups of perpendicular lines that reflect the presence of stairs. No such periodic lines are typically present when traversing a single floor. One simple solution to this problem is identifying individual stairs using the Hough

transform. Another solution includes the use of a Gabor filter at the spatial frequency of the steps [51] and further searching for candidate lines for the individual stairs in the image using a combination of the Hough transform and the surrounding optical flow [52]. These methods have proven successful for robotic applications but because the target of this algorithm is real-time implementation on a smartphone, we have opted for the less computationally intensive approach of using Canny edge detection and searching for perpendicular and parallel lines within the image using the Hough transform. An example is found in Fig 4.1.

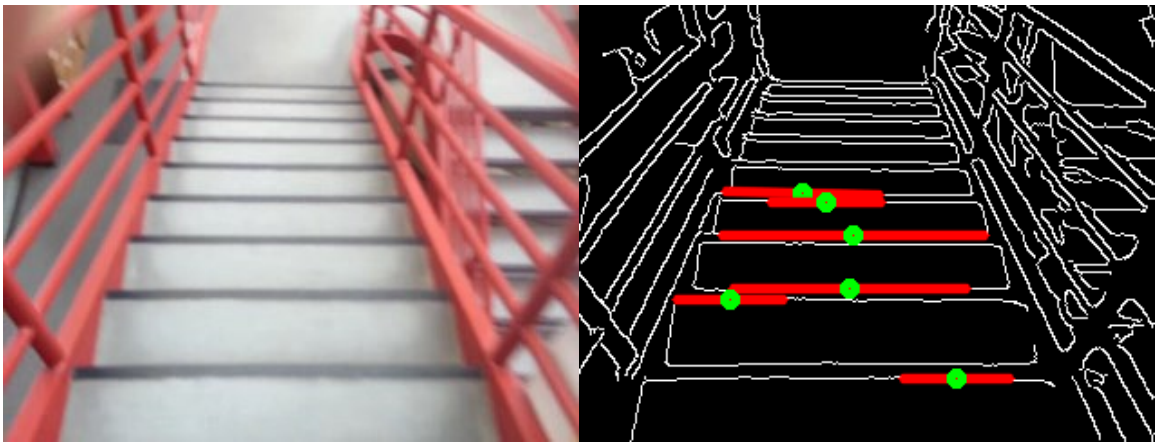


Fig 4.1. Canny edge detection and the Hough transform used for stair detection for a device undergoing motion on stairs.

4.3.1 Canny Edge Detection

The Canny edge detector aims to outline the edges in a grayscale image [53]. First described in 1986, it was designed as an optimal detector and, for this reason, is still used in modern image processing applications. A simplified overview of the algorithm is as follows, summarizing the description in [53].

Edge detection operates in multiple stages [53]. The first stage applies Gaussian smoothing to the image in order to remove noise [53]. Smoothing is accomplished in an identical manner to that described in the smoothing sub-stage of the pre-processing section described in Chapter 3. The second stage calculates the horizontal, vertical and diagonal gradients of the image in order to find regions of high intensity variations indicative of an edge [53]. The third and fourth stages serve to reduce the number of edges that are reported by the algorithm for each edge actually present in the image [53].

4.3.2 Hough Transform

The edge-detected image is fed into the Hough transform for line detection. The Hough transform is a popular algorithm for detection of arbitrary shapes within an edge-detected image [54]. Because the computational complexity increases exponentially with the number of shape parameters, the algorithm runs most efficiently when used to detect simple parameterized shapes such as lines. Furthermore, the Hough transform is robust against the presence of noise and can be used to detect incomplete shapes. The original algorithm is described in [54] and a brief overview of the algorithm follows, following the description found in [55].

In this discussion, the edge detected image will be referred to as the *xy*-plane. As we are detecting lines, we aim to detect a shape in the *xy*-plane that follows the slope-intercept equation [55]:

$$y = mx + b \tag{60}$$

where m is the slope of the line and b is the y-intercept.

In this scenario, it is more mathematically convenient to work in polar geometry. The polar representation of a straight line is given by [55]:

$$\rho = x \cos\theta + y \sin\theta \tag{61}$$

Each point in the xy-plane maps to a sinusoid in the $\rho\theta$-plane [55]. The value of θ ranges from 0 to ±90 degrees whereas ρ ranges from $\pm N\sqrt{2}$ for an image of size NxN [55].

We construct an accumulator cell matrix $A(\theta,\rho)$ where $\rho \in [\rho_{\min}, \rho_{\max}]$ and $\theta \in [\theta_{\min}, \theta_{\max}]$ [55]. The width of the accumulator partitions $d\rho$ and $d\theta$ must be chosen so as to give acceptable precision while taking into account the size of the accumulator matrix [55].

For each point (x_k, y_k) in the xy-plane, we increment θ_i across its range, incrementing by $d\theta$, and solve equation (61) for ρ [55]. Rounding the value of ρ to its closest bin value, each bin being separated by $d\rho$, we increment the value of the accumulator cell $A(i,j)$ if θ_i results in ρ_j [55]. In other words, the value of $A(i,j)$ gives the number of points in the xy-plane that lie on the line characterized by parameters (θ_i,ρ_j). This value is then thresholded [55] to find lines of sufficient length to be indicative of stairs.

4.3.3 Stairs Direction

Having identified that the device is undergoing motion on stairs in the texting use case, the direction of travel (up or down) is determined using different techniques depending on the current pitch, roll, and misalignment values. For device misalignments around 0 or 180 degrees and pitch angles around zero degrees (i.e. phone held primarily facing the ground in a portrait orientation) or misalignments around 90 or -90 and roll angles around 0 or 180 degrees (i.e. phone held primarily facing the ground in a landscape orientation), the images can be examined for multiple shrinking lines that arise within the stair borders due to the perspective of projecting a three-dimensional downward stairwell onto the two-dimensional image plane. Therefore, a positive in this test can indicate downwards motion while a negative indicates motion upwards. Another

method is to examine the number of lines detected, under the assumption that a device travelling down a staircase will suddenly detect a large number of lines in its field of view as opposed to when the device is travelling up and only a few stair edges are in the camera's view.

Yet another method to determine the direction of motion in the stairs texting use case is to examine the ratio of the leveled horizontal and vertical components of the aggregated optical flow after integration over a small moving window. As there is a decreased distance of the observed ground plane relative to the camera when travelling upwards on stairs, the pixels undergo movement at a faster relative velocity and a greater flow ratio is seen in the axis that aligns with the direction of motion when travelling up stairs than that seen when travelling down. There is a clearer boundary between the flow ratio observed in each of the two directions when the device is held at greater pitch angles. This is the approach used in this work.

4.3.4 Tiled Floor

An important aspect of correctly classifying the stairs texting use case is the ability to distinguish between floor patterns that are indicative of stairs and those indicative of tiles (a common floor pattern). An example of a tiled pattern is found in Fig 4.2. Comparing Fig 4.1 and 4.2, we see that motion on a tiled surface exhibits a greater number of lines parallel to the direction of motion. Performing a check on the number of parallel lines found can distinguish the two cases.

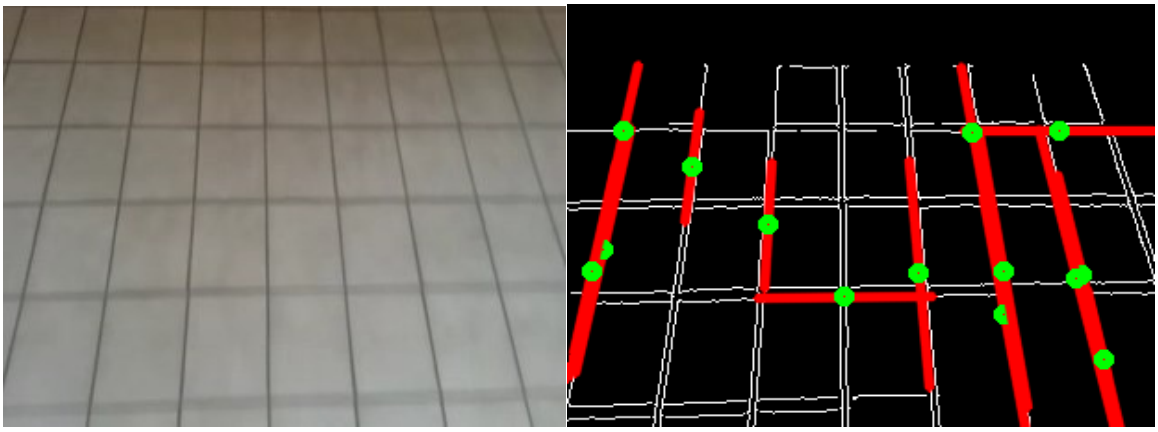


Fig 4.2. Canny edge detection and the Hough transform used for stair detection for a device undergoing motion on a tiled surface.

4.4 Calling Use Case – Single Floor vs. Stairs

The single floor calling use case can be distinguished from the stairs calling use case by examining the ratio of the leveled horizontal and vertical components of the aggregated optical flow after integration over a small window in time. Theoretically, if the user moved in equal amounts in the horizontal and vertical direction as he/she traversed each step, the ratio of the vertical to horizontal flow over a small window in time should equal unity. Practically, there are fluctuations around unity dependent on the characteristics of a specific user's gait. As such, the ratio of the magnitude of vertical to horizontal flow after integration over a small window is expected to yield a value with a window near unity in the stairs calling use case.

4.4.1 Stairs Direction

In the case of the stairs calling use case, the sign of the vertical component of the aggregated flow at that time is used to determine the direction of motion (up or down).

Chapter 5: Results for Device Angle Calculation Module

The device used for image acquisition was the Samsung Galaxy S3 running Android 4.0. The Samsung Galaxy S3 features tri-axial gyroscopes, tri-axial accelerometers, tri-axial magnetometers, barometer, GNSS chipset, as well as forward facing and rear facing cameras. The camera on the back of the phone (i.e. facing away from the screen) was used to capture video in specific scenarios. The video files representing the various test cases were then processed offline using the algorithms described in Chapters 3 and 4. The algorithms were implemented in C++. A description of each test and the corresponding results are as follows.

5.1 Availability and Static Detection

The first test was 40 seconds in duration and consisted of four 10-second segments. From 0-10 seconds, the device was held statically in front of a wall of uniform color. From 10-20 seconds, the device was in motion in front of the same wall. From 20-30 seconds, the phone was held static while facing the ground in an office environment. From 30-40 seconds, the device was in motion facing the same floor.

Examining the values of the aggregated intensity gradient and MSSIM in Fig. 5.1, we see that both parameters are nearly identical for the static and moving cases when the camera is facing a uniform, featureless surface. No meaningful optical flow information is obtained in these cases and the vision module is deemed unavailable in this context. With the device held statically facing a feature-rich environment, we see a significantly higher (largely constant) value of the aggregated intensity gradient while the MSSIM stays close to 1 as in the previous case. The intensity gradient begins to fluctuate around a large value while the MSSIM exhibits fluctuations around a value

smaller than 1 for the case of a moving device in a feature-rich environment. Thresholding the values in Fig. 5.1 gives the device status as shown in Fig. 5.2.

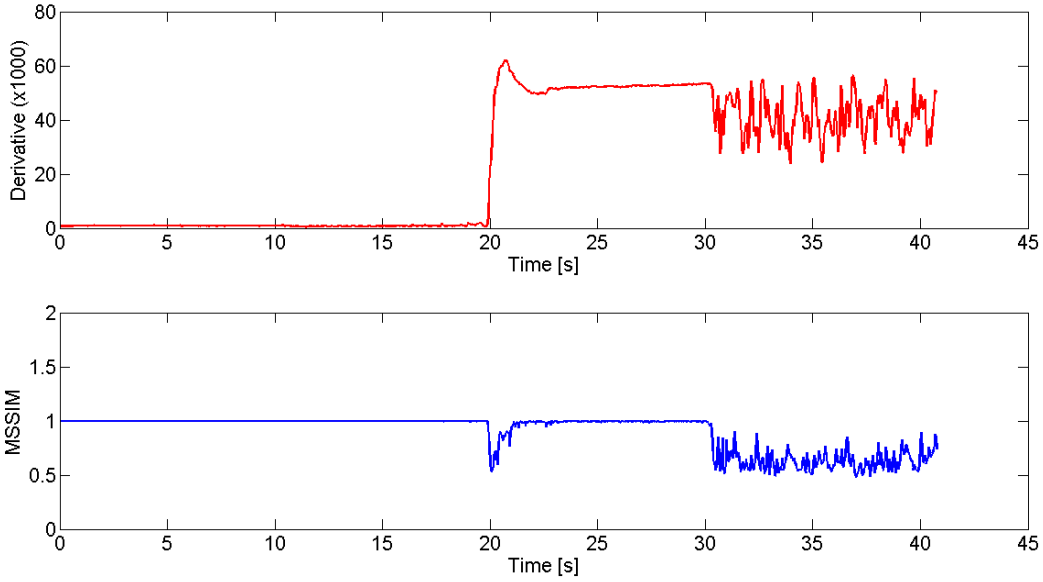


Fig. 5.1. Spatial derivative and MSSIM values.

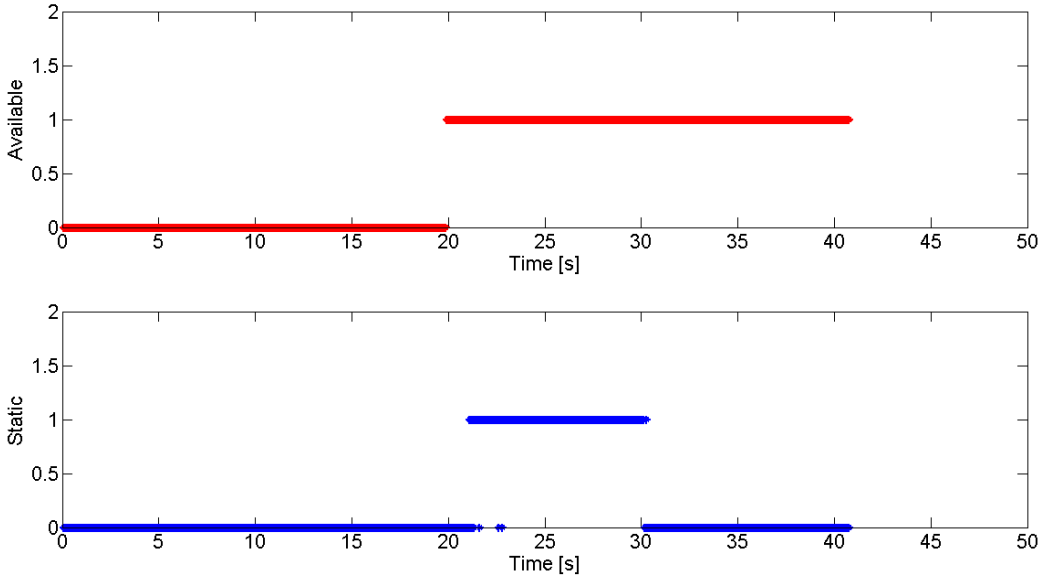


Fig. 5.2. Availability and static indicators.

5.2 Device Angle Calculation

Having determined that the device is in a feature-rich environment and is no longer static, we must differentiate between meaningful motion and stationary fidgeting. From the check on availability and static mode, we know that the images present valid optical flow information.

5.2.1 Texting Use Case Device Angle

The texting use case is one of the most common ways in which a handheld device is used. In this use case, the user may be sending a message, checking a map, or reading an article. To test the algorithm, the user was asked to walk forward with the device and rotate the device counter clockwise in 90 degree increments after approximately 4 seconds in each orientation. Figure 5.3 shows the camera view when the device is in the texting use case at an angle of zero degrees.

Fig. 5.3. Camera view when device is in the texting use case at an angle of zero degrees.

The aggregated flow of pixels from the flow map is shown in Fig 5.4. At a device angle of zero degrees from 0 to 4s, a strong flow in the positive x-direction is observed with minor oscillations

occurring in the y-direction. This is to be expected as there is significant pixel flow from the left to the right side of the image and only a small amount of motion vertically due to the disturbances arising from the user's hand movements and steps. Rotating the device at t = 4 s to the 90 degree orientation resulted in a strong flow in the y-direction with a reduced, noisy x-flow. Similar behavior (albeit vice versa) was observed for the remaining orientations of 180 and -90 degrees, starting at t = 8 s and t = 12 s. The arctangent of the aggregate flows gives the estimate of the device angle with respect to the user as shown in Fig. 5.5. Though the results are quite noisy, it is clear that the device angle estimates are centered around the four orientations of 0, 90, 180, and -90 degrees. Smoothing the aggregate flows of Fig. 5.4 using a moving average window size of 15 frames before calculating the device angle resulted in the smoothed angle estimates given in Fig. 5.6. It is evident from this figure that the device angle is changing by 90 degrees approximately every four seconds.

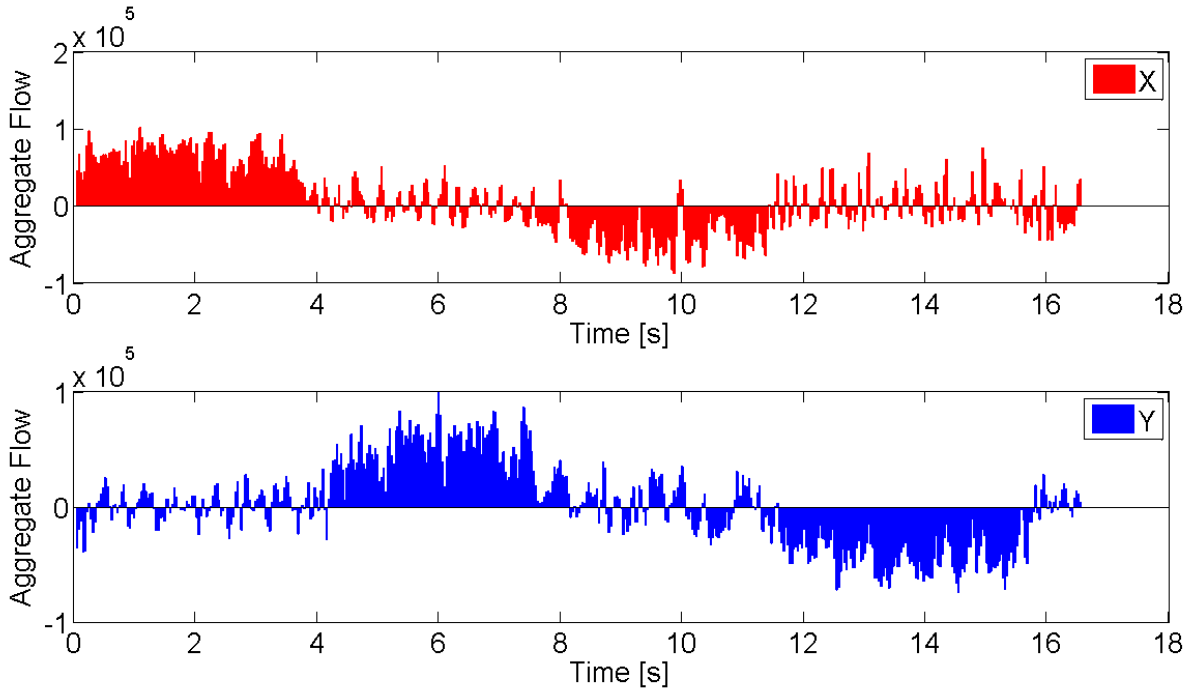


Fig. 5.4. X- and y-components of the aggregate flow for the texting use case trajectory.

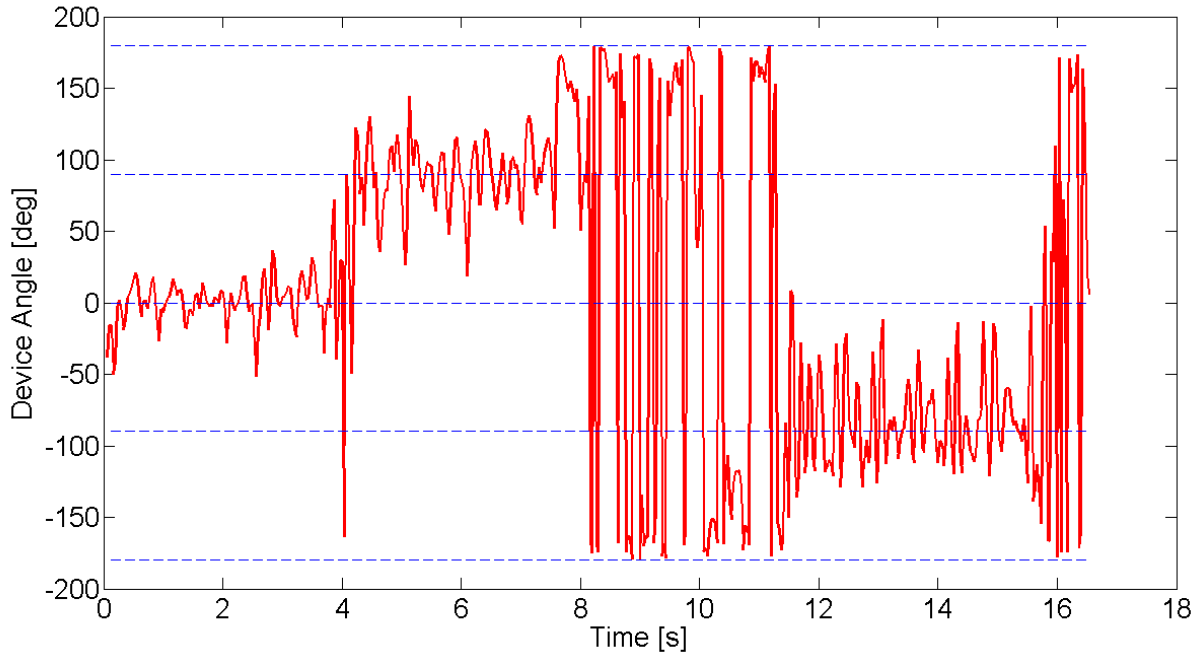


Fig. 5.5. Device angle computed using the arctangent of the aggregated x- and y-flow.

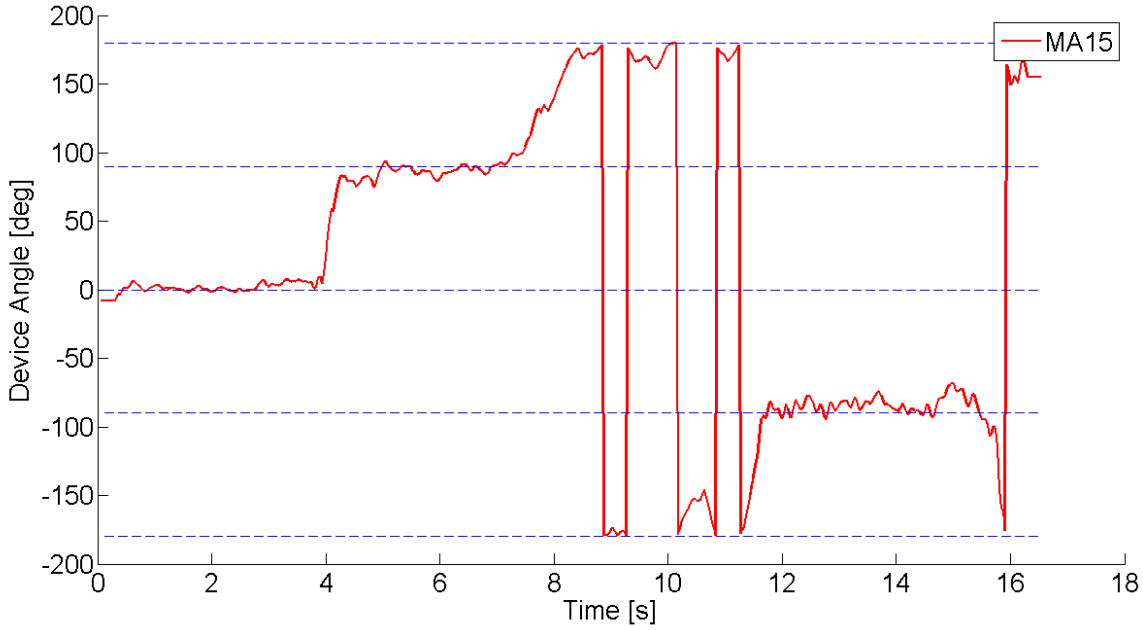


Fig. 5.6. Device angle computed using the arctangent of the smoothed x- and y-components of the aggregate flow (smoothed over 15 frames, equivalent to 0.5 seconds at 30 fps).

5.2.2 Dangling Use Case Device Angle

Fig. 5.7 shows a sample image captured at equilibrium position when the device undergoing dangling at an angle of zero degrees. Equilibrium position is defined as the time when the arm is in line with the body. Three separate tests were conducted to show the various signals when the device was dangling with a known misalignment value of zero, 180, and -90 degrees at equilibrium position. The misalignment value of +90 degrees, while accounted for in the algorithm design, was ignored in testing as holding the device at such an angle resulted in the camera being completely blocked by the user's hand. Hence, this was deemed a non-practical use case to address with the described technique.

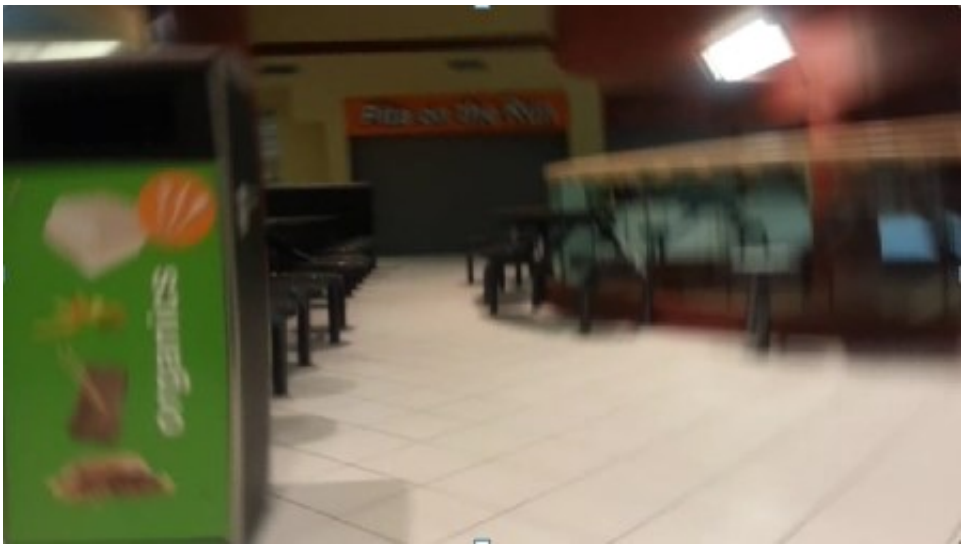


Fig. 5.7. Camera view when device is in the dangling use case at equilibrium position.

5.2.2.1 Dangling – Zero Degrees

The test was conducted with the camera facing away from the body and pointing towards the direction of the user's motion. Due to the nature of the dangling motion, a strong oscillatory flow was expected in the x-direction. While some oscillatory flow is expected in the y-direction, the

amount of translation undergone in the vertical is significantly smaller than that undergone horizontally and greater periodicity is to be expected in the x-flow than in the y-flow. Fig. 5.8 shows the values of the aggregate flow for the dangling trajectory.

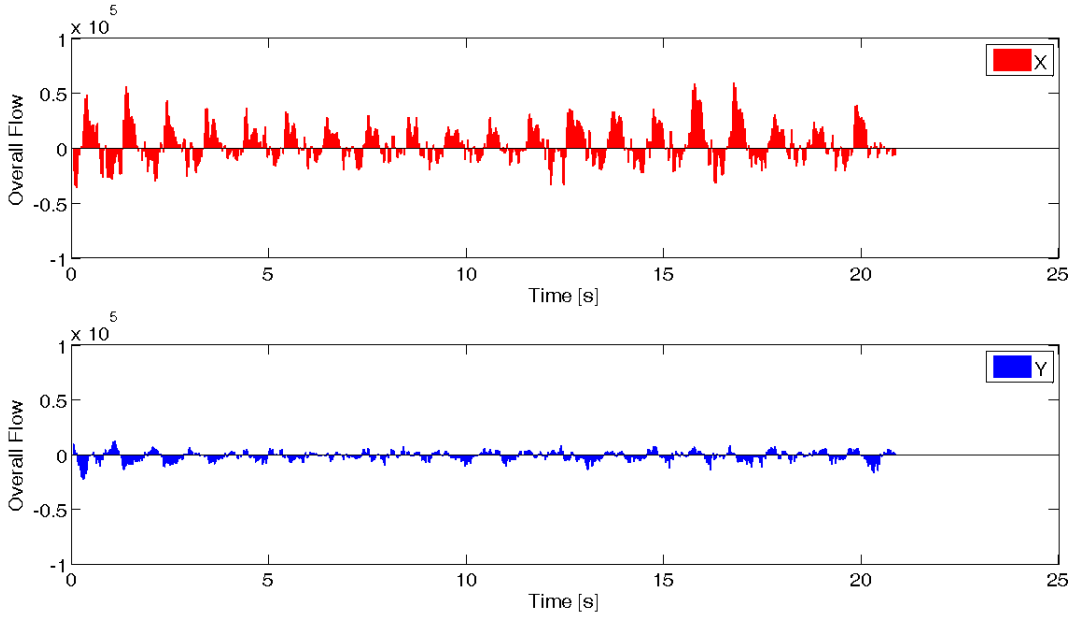


Fig. 5.8. X- and y-components of the aggregate flow for the dangling trajectory at a device angle of zero degrees.

By binary thresholding the x- and y-components of the aggregate flow shown in Fig. 5.8, we obtain a signal that resembles a periodic square pulse train with values of +1 and -1 when the aggregate flows are positive and negative, respectively. Figure 5.9 shows the results of differencing the binary thresholded flow signal. Fig. 5.11 shows the magnitude spectrum of the differenced signal in Fig. 5.9 and gives an indication of the degree of the periodicity for the x- and y-components of the optical flow. Inspecting the magnitude spectrum in Fig 5.10, we see a clear peak that exceeds the detection threshold value at a frequency of $f = 0.96$ Hz. This value represents the dangling frequency and the fact that the peak is larger for the x-component than the the y-

component confirms that the device is undergoing dangling at a misalignment angle of either zero or 180 degrees. Using the state machine outlined in Fig 3.8 on the binary differenced signal of Fig 5.9 yields an array outlining when the device is in the positive and negative half-cycles of the dangling motion. The PHC/NHC state array can be found in Fig 5.11. Integrating the flow values of Fig 5.8 in each of the half-cycle states determined in Fig 5.11 allows for a comparison of the overall translation undergone in each half cycle of the dangling motion (and is shown in Fig 5.12). Using the values of the integrated flows in each half cycle and equation (50), we obtain the dangling device angle shown in Fig 5.13.

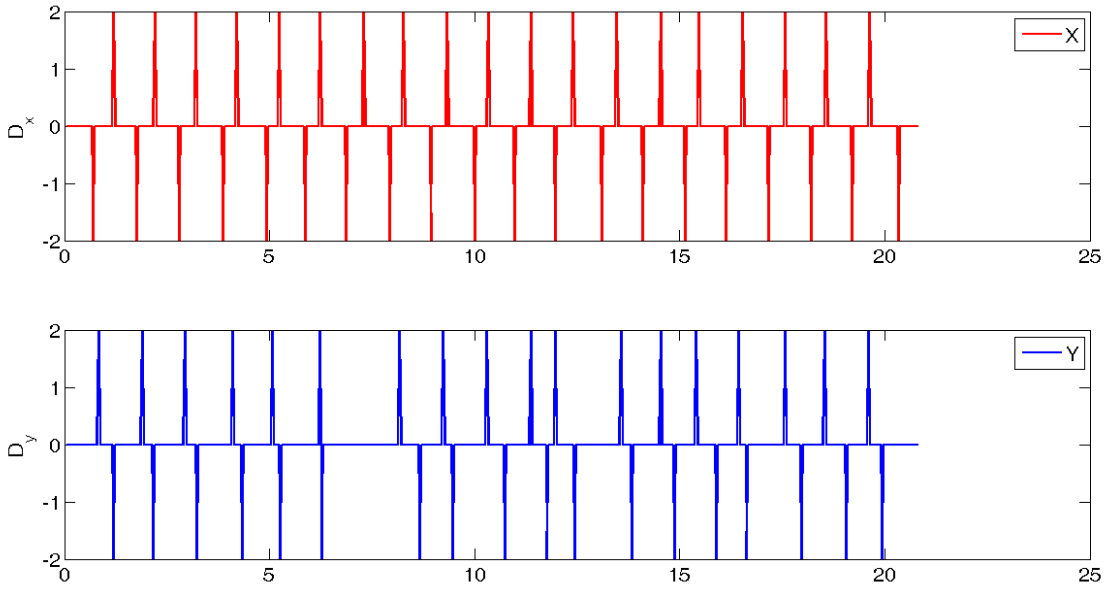


Fig. 5.9. Differencing of the binary thresholded aggregate flow for a device undergoing dangling at an angle of zero degrees. Each spike indicates a change in dangling direction.

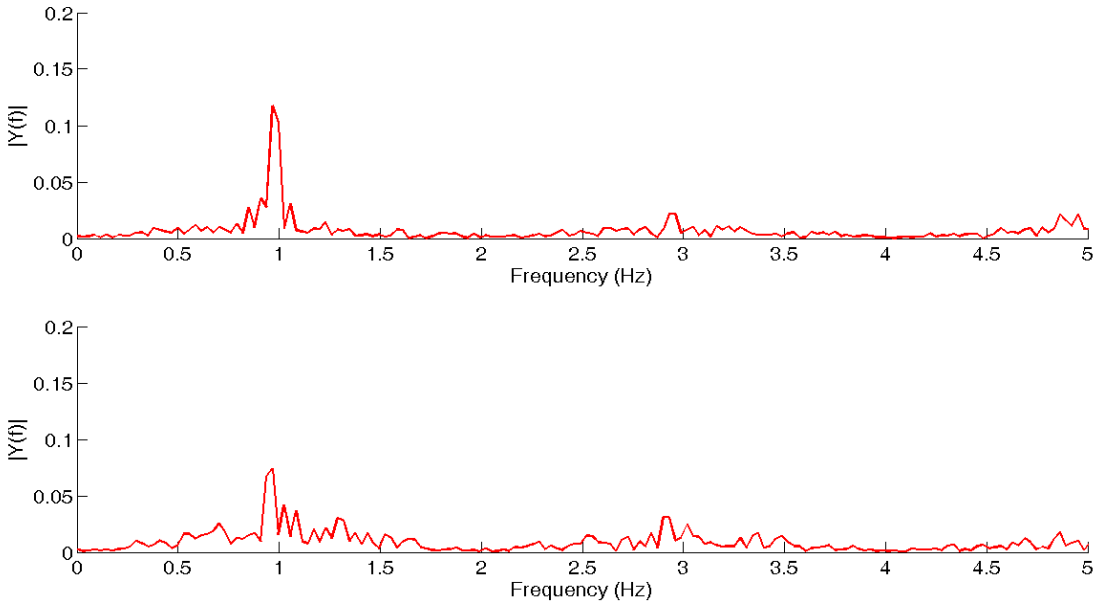


Fig. 5.10. Magnitude spectrum of the differenced binary thresholded aggregate flow for a device undergoing dangling at an angle of zero degrees. The peak indicates the dangling use case and occurs at the dangling frequency.

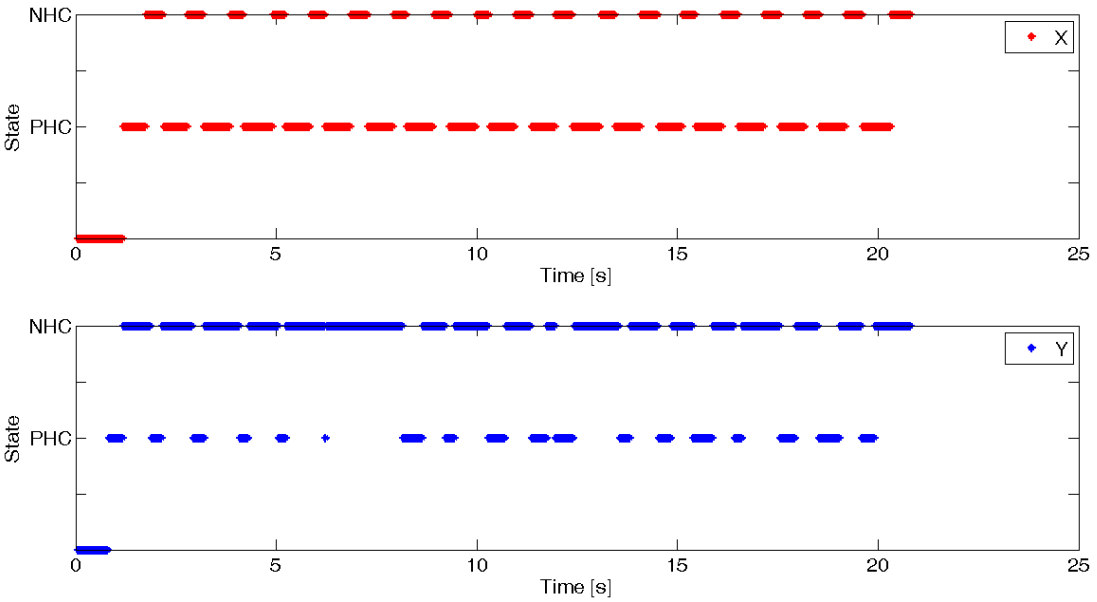


Fig. 5.11. Separation of the dangling motion into positive and negative half-cycles for a device undergoing dangling at an angle of zero degrees.

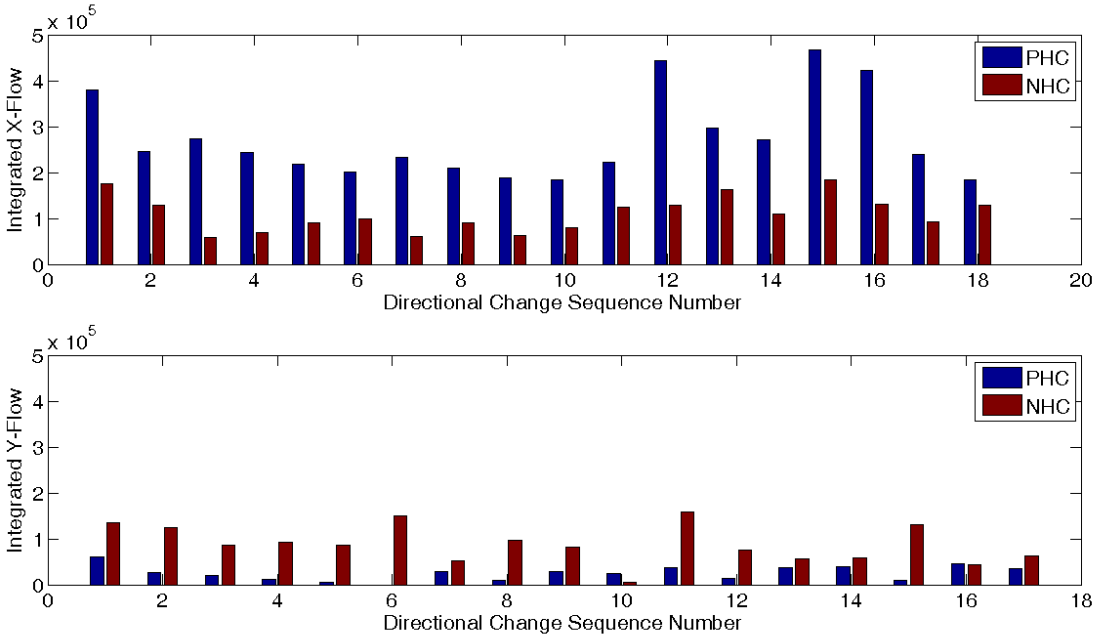


Fig. 5.12. Integrated x- and y-flows during the positive and negative half-cycles of a device undergoing dangling at an angle of zero degrees.

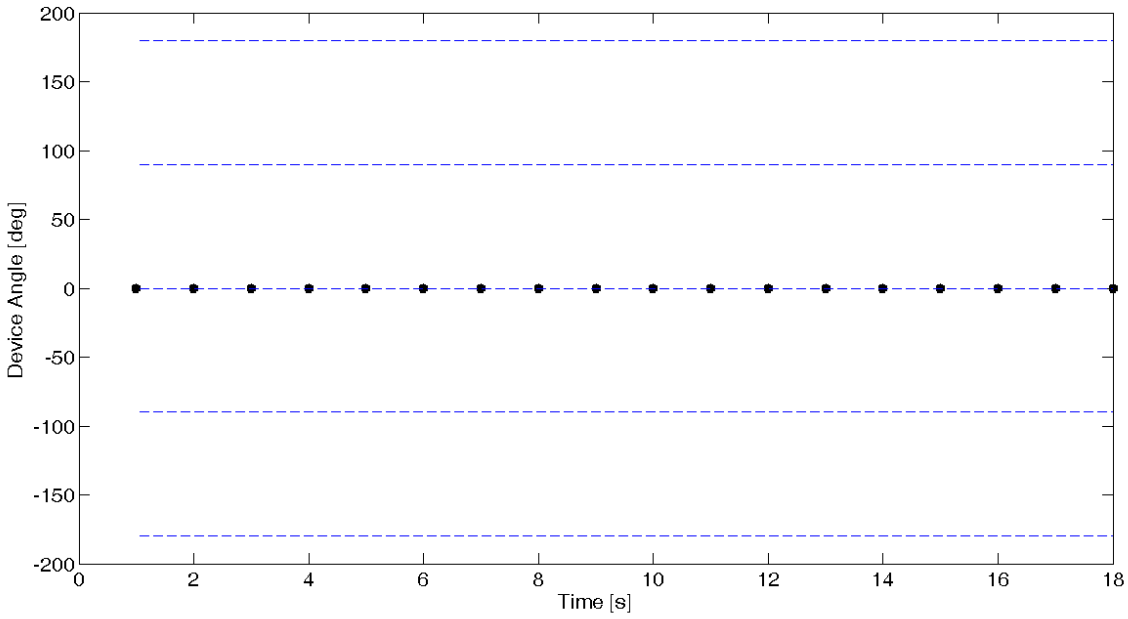


Fig. 5.13. Device angle at equilibrium position for a device undergoing dangling at an angle of zero degrees.

5.2.2.2 Dangling – 180 Degrees

An identical test as that described above was carried out with the device camera facing out and against the direction of the users motion. The plots of Fig. 5.8-5.13 were regenerated for a device angle of 180 degrees and can be found in Fig. 5.14-5.19, respectively. The main difference expected from this series of plots as compared to those for the zero degree misalignment case is that of Fig. 5.18, the integrated flows over each half-cycle. In the 180 degree misalignment case, we expect to see greater x-flow in the negative half-cycles than in the positive half-cycles.

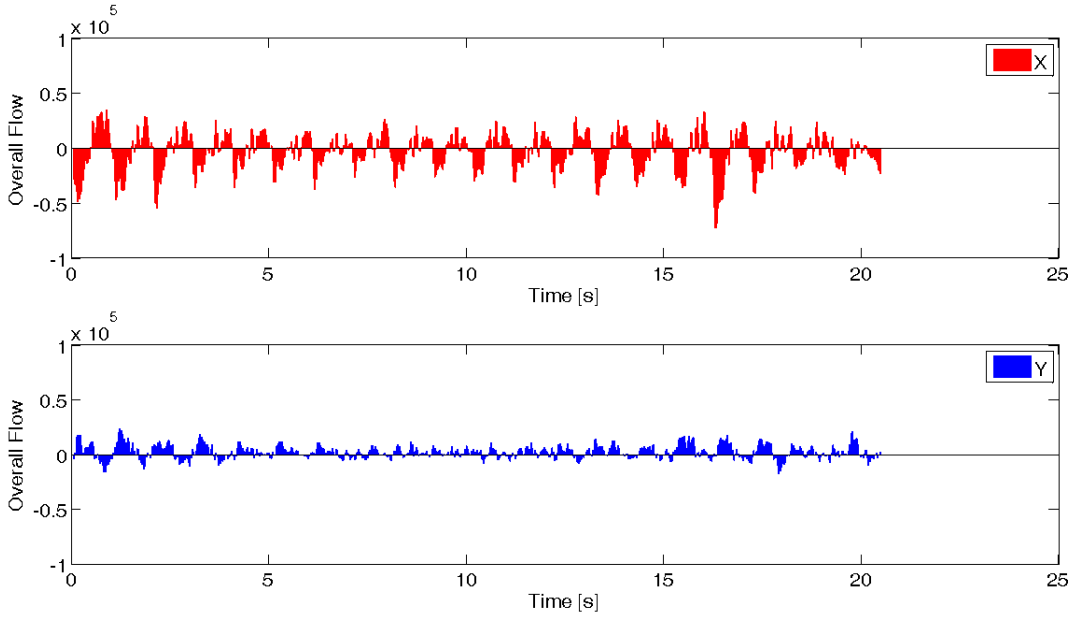


Fig. 5.14. X- and y-components of the aggregate flow for the dangling trajectory at a device angle of zero degrees.

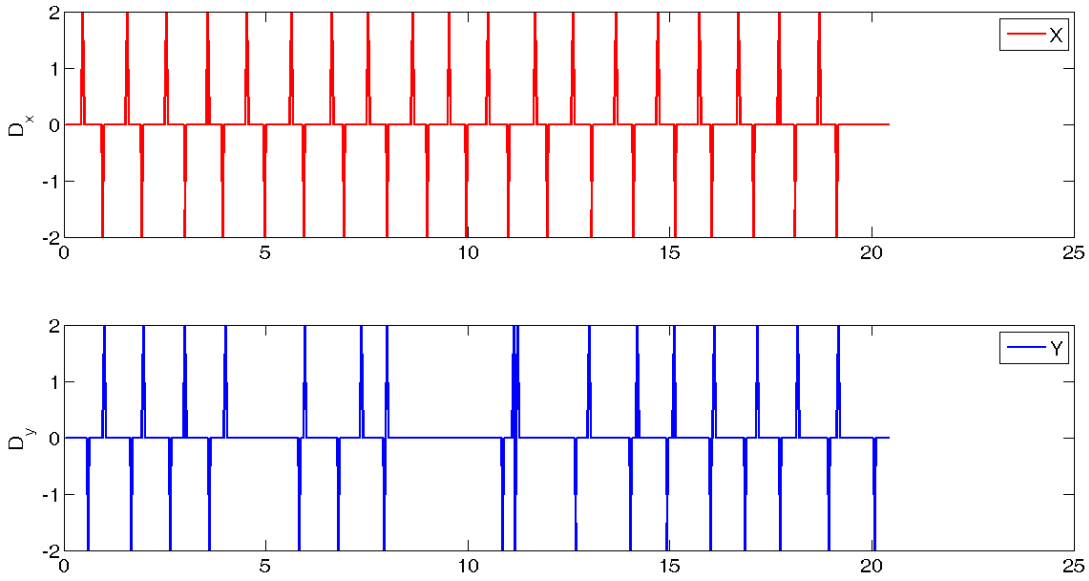


Fig. 5.15. Differencing of the binary thresholded aggregate flow for a device undergoing dangling at an angle of zero degrees. Each spike indicates a change in dangling direction.

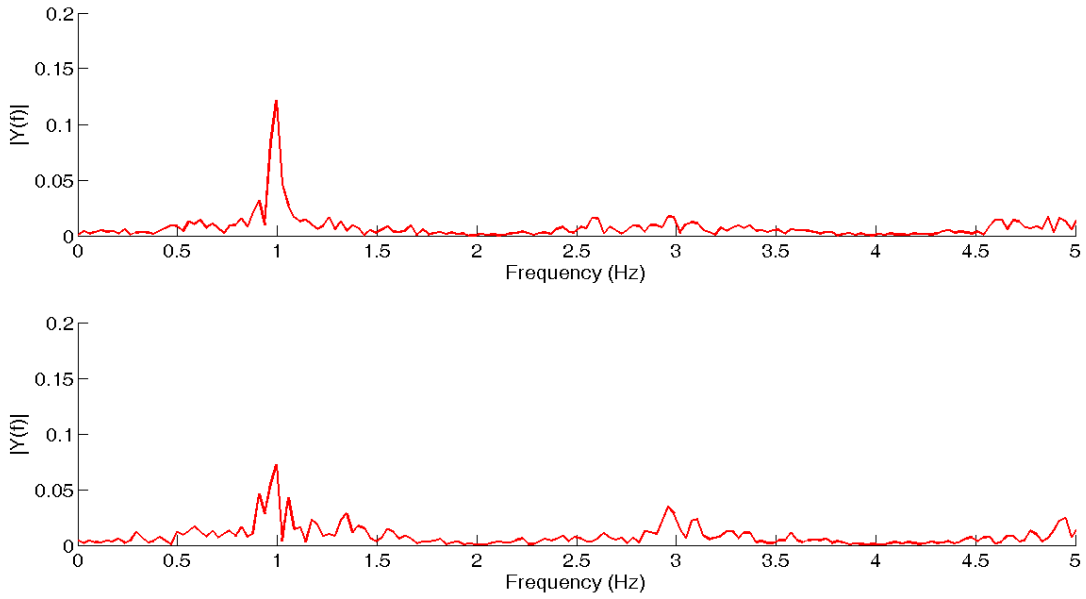


Fig. 5.16. Magnitude spectrum of the differenced binary thresholded aggregate flow for a device undergoing dangling at an angle of 180 degrees. The peak indicates the dangling use case and occurs at the dangling frequency.

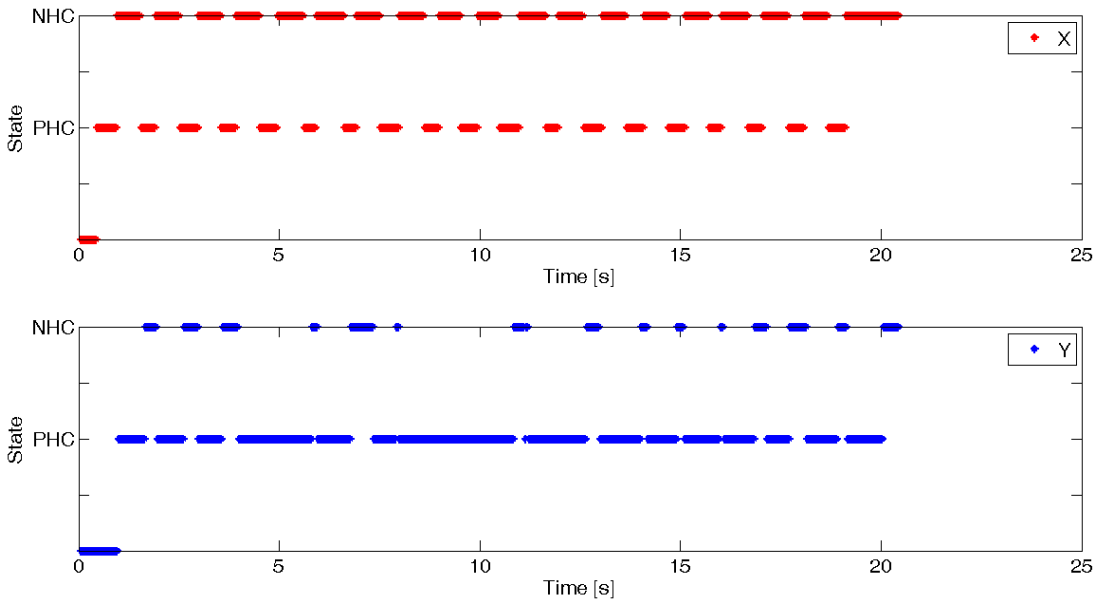


Fig. 5.17. Separation of the dangling motion into positive and negative half-cycles for a device undergoing dangling at an angle of 180 degrees.

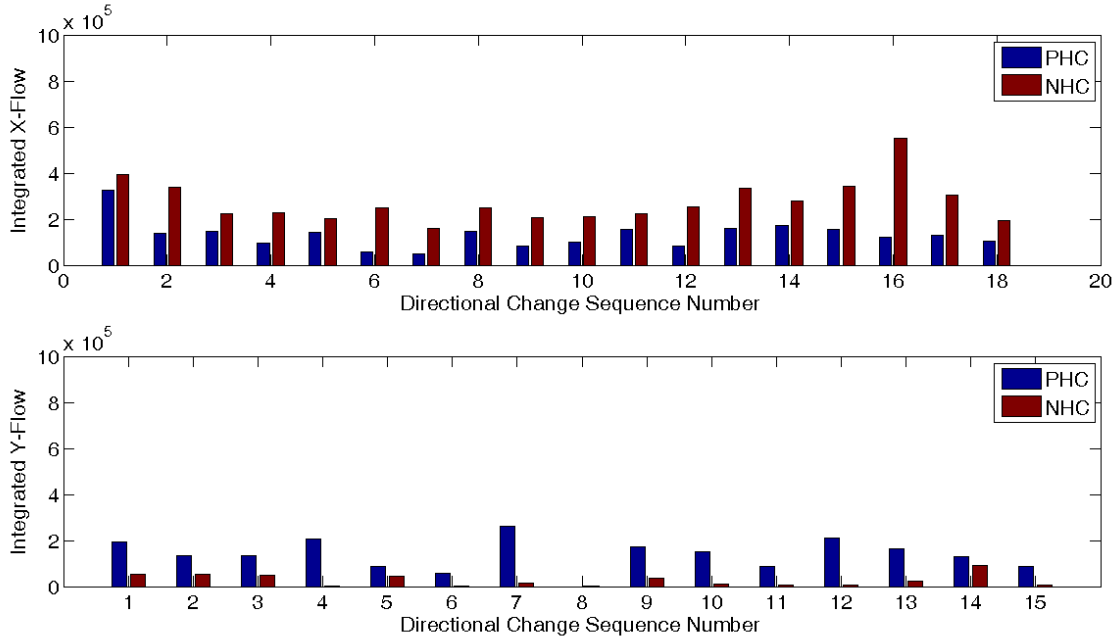


Fig. 5.18. Integrated x- and y-flows during the positive and negative half-cycles of a device undergoing dangling at an angle of 180 degrees.

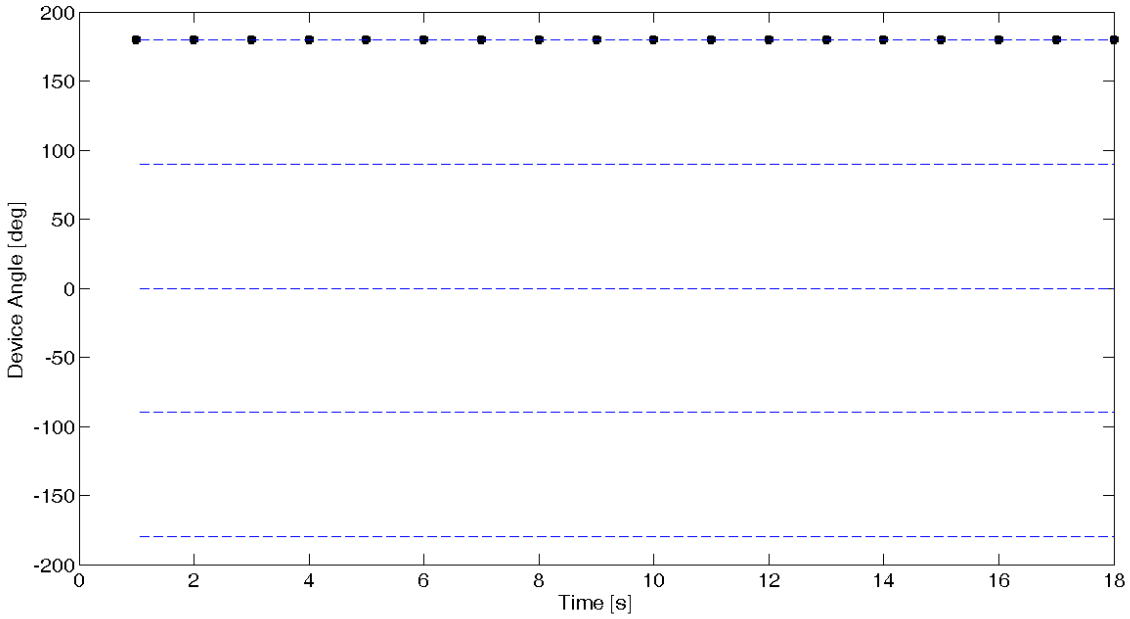


Fig. 5.19. Device angle at equilibrium position for a device undergoing dangling at an angle of 180 degrees.

5.2.2.3 Dangling – -90 Degrees

An identical test as that described above was carried out with the device camera facing out and down towards the ground. The plots of Fig. 5.14-5.19 were regenerated for a device angle of -90 degrees and can be found in Fig. 5.20-5.25. The main difference between the current and previous set of plots can be found in Fig. 5.20 and 5.22 where stronger periodicity can be seen in the y-flow signal.

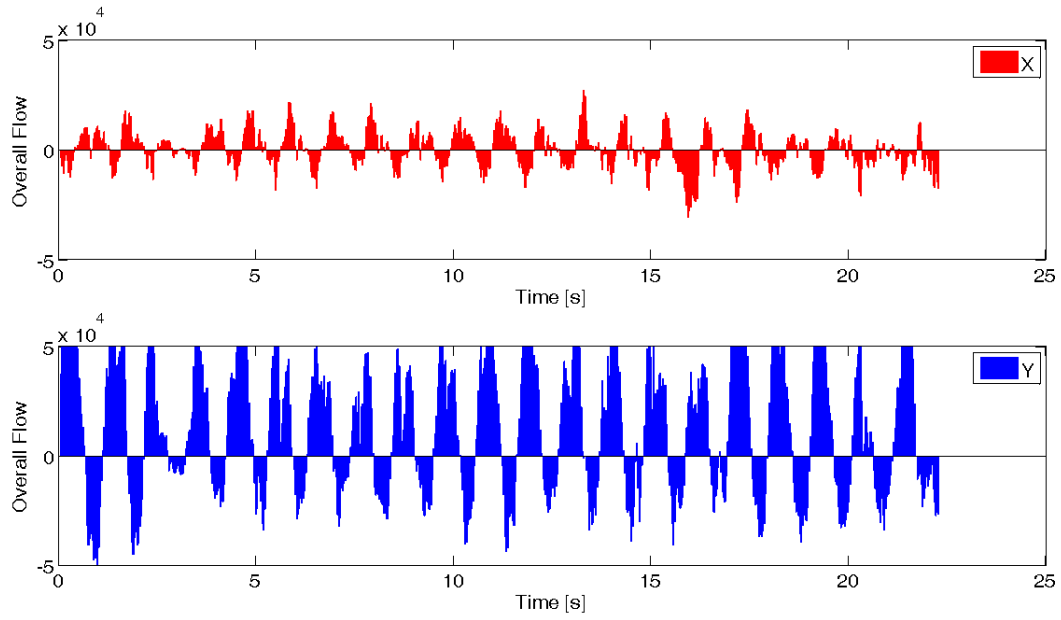


Fig. 5.20. X- and y-components of the aggregate flow for the dangling trajectory at a device angle of -90 degrees.

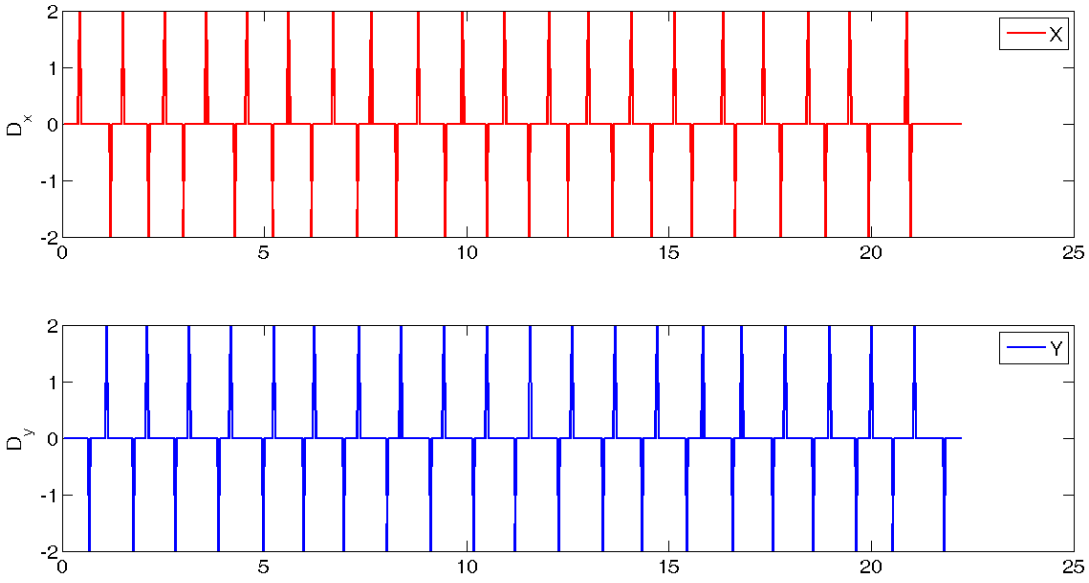


Fig. 5.21. Differencing of the binary thresholded aggregate flow for a device undergoing dangling at an angle of -90 degrees. Each spike indicates a change in dangling direction.

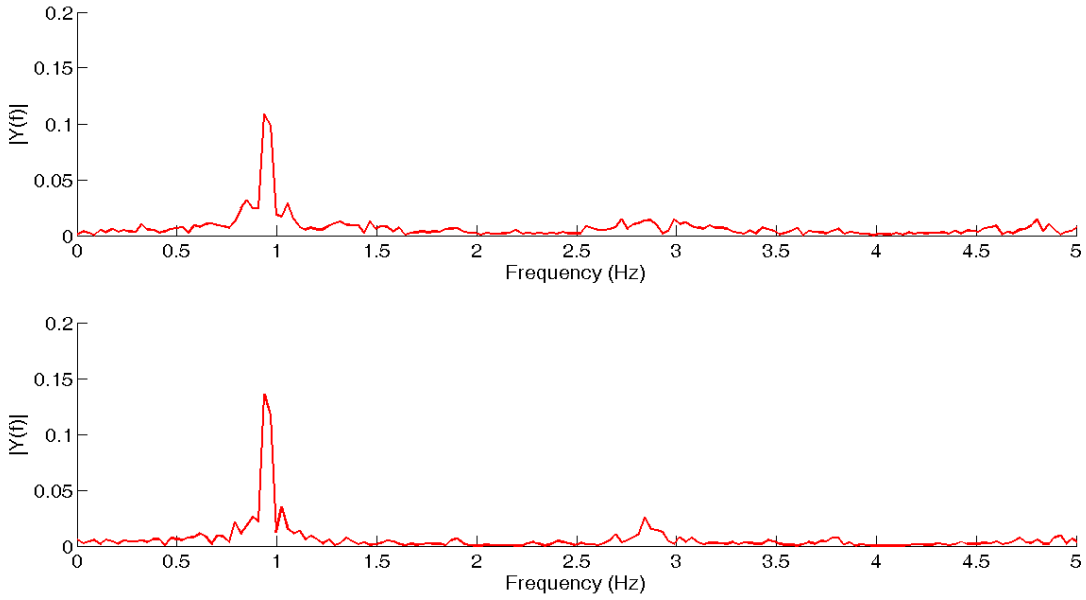


Fig. 5.22. Magnitude spectrum of the differenced binary thresholded aggregate flow for a device undergoing dangling at an angle of -90 degrees. The peak indicates the dangling use case and occurs at the dangling frequency.

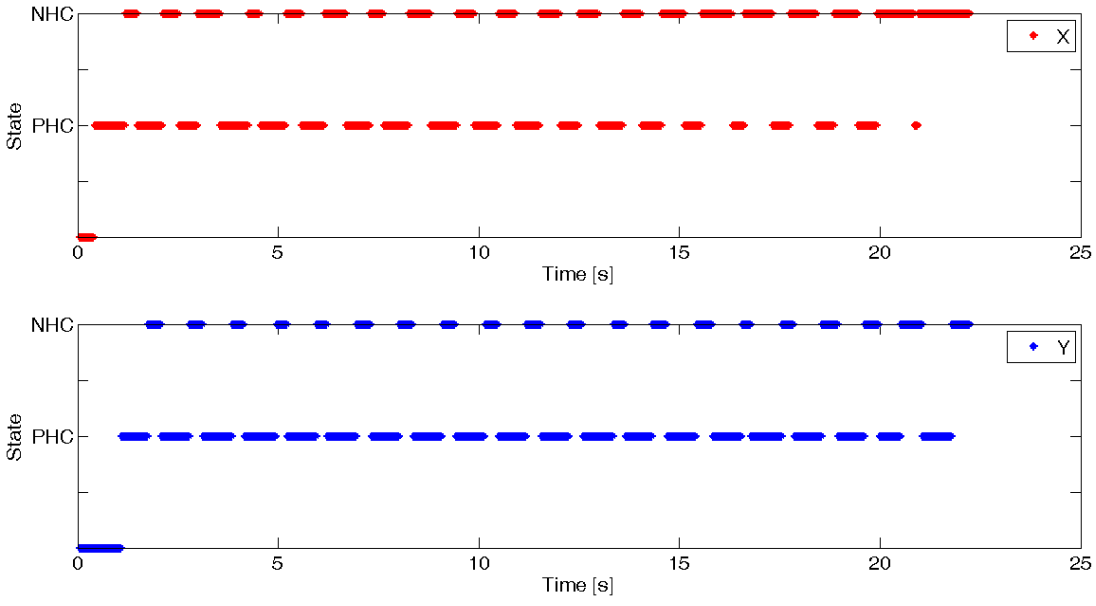


Fig. 5.23. Separation of the dangling motion into positive and negative half-cycles for a device undergoing dangling at an angle of -90 degrees.

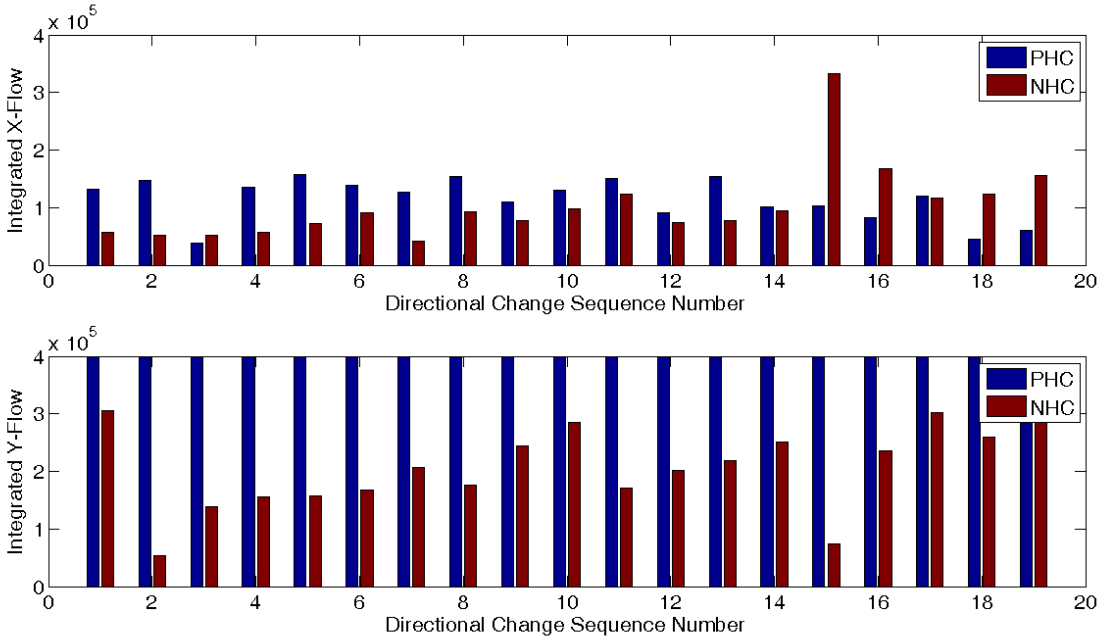


Fig. 5.24. Integrated x- and y-flows during the positive and negative half-cycles of a device undergoing dangling at an angle of -90 degrees.

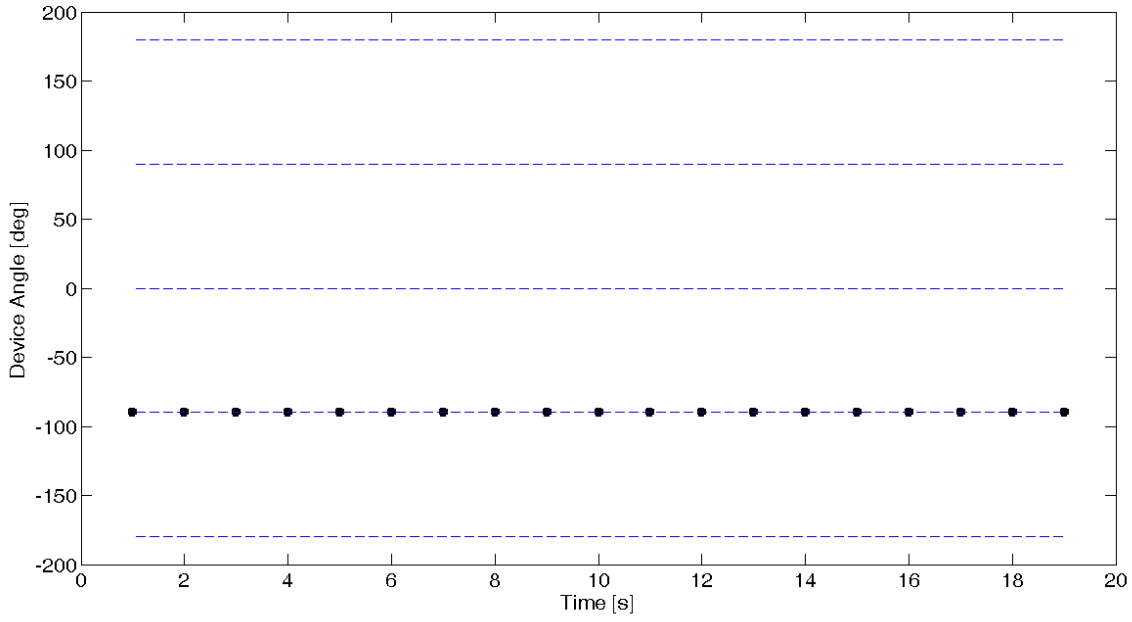


Fig. 5.25. Device angle at equilibrium position for a device undergoing dangling at an angle of -90 degrees.

Chapter 6: Results for Context Classification Module

The device used in context classification testing was the same one used in the previous chapter for the device angle module. A description of each test and the corresponding results are as follows [44].

6.1 Fidgeting

Two data sets were collected in order to assess fidgeting detection. For the first dataset, a user held the phone in the texting use case while walking indoors in a straight line for twenty seconds. At approximately ten seconds through the test, the phone orientation was changed by 90 degrees. Fig. 6.1 shows the components of the aggregated flow, Fig 6.2 shows the calculated device angle, and Fig 6.3 shows the variance of the device angle. It is worth noting that Fig 6.3 shows a device angle variance that is consistently below 20 degrees. The second dataset involved the user engaging in stationary fidgeting (i.e. actions in which the device undergoes motion that is not meaningful in the context of navigation such as texting, playing a game, using the camera, etc.) for thirty seconds. Fig 6.4 shows the variance in the calculated device angle associated with the fidgeting use case and it can be seen that the variance hovers around 90 degrees. Comparing Fig 6.3 and Fig 6.4, we see that a clear separation exists between meaningful straight line motion and fidgeting.

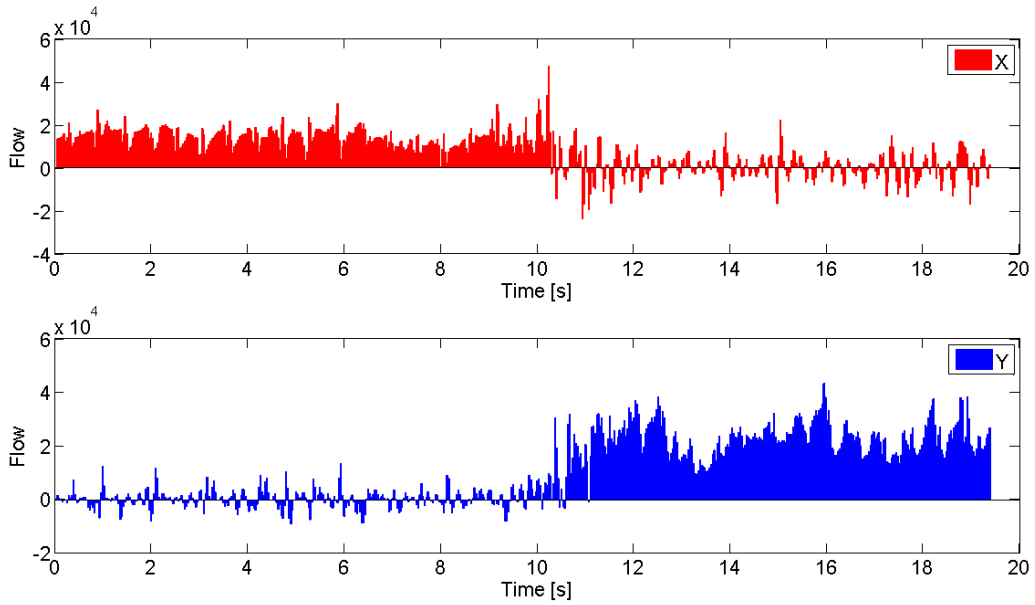


Fig. 6.1. X- and y-components of aggregated flow for a single floor texting context. An orientation change from portrait to landscape occurs at a time of approximately t = 10 seconds.

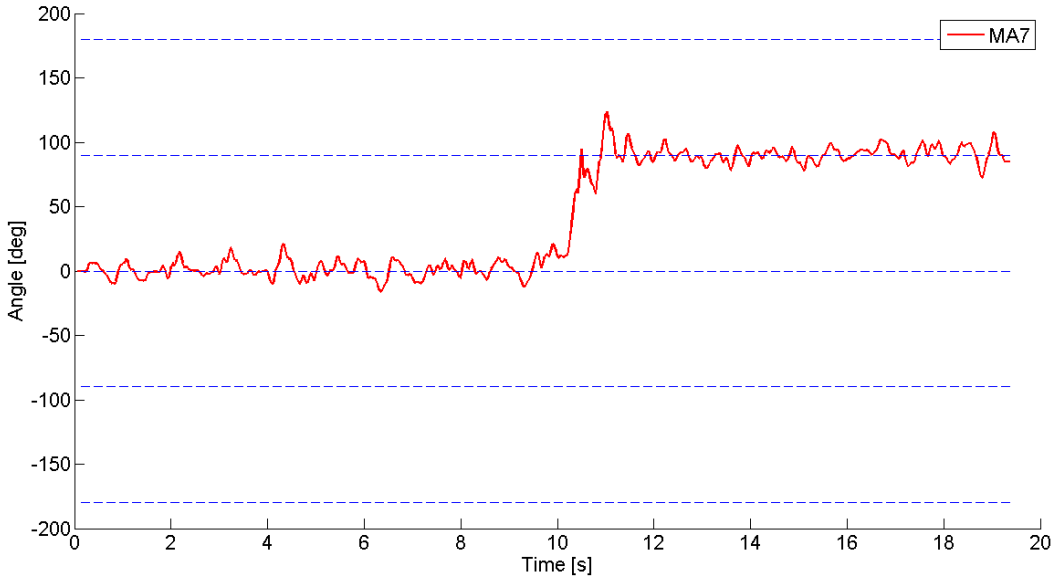


Fig. 6.2. Device angle calculation for a single floor texting context. An orientation change from portrait to landscape occurs at a time of approximately t = 10 seconds.

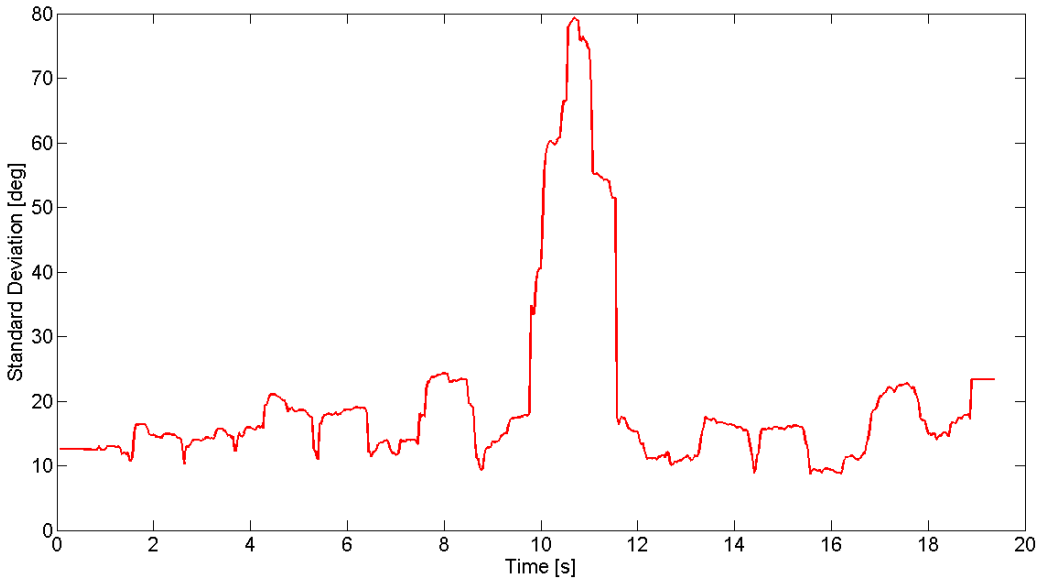


Fig. 6.3. Device angle standard deviation over a moving window of 30 frames (1 second) for a single floor texting context. An orientation change from portrait to landscape occurs at a time of approximately $t = 10$ seconds.

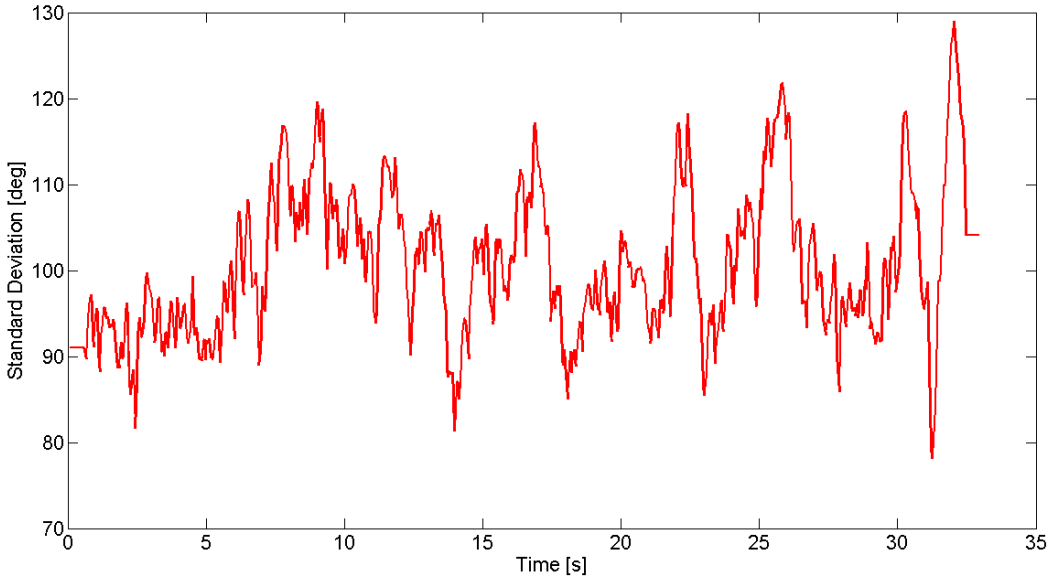


Fig. 6.4. High standard deviation values for the device angle are indicative of fidgeting.

6.2 Texting Use Case – Single Floor vs. Stairs

The single floor and stairs texting context can be differentiated by examining the number of perpendicular and parallel lines found in each frame. These values can be found in Fig. 6.5 for a 45-second data set in which the user walked up two flights of stairs and back down. In between each flight was a level floor where the user would make a heading change of 180 degrees before continuing to the next flight of stairs. The device was held in a landscape orientation. The instantaneous values of the number of perpendicular and parallel lines found in Fig. 6.5 indicate a noticeable trend which is more pronounced after integration over a 1-second window (see Fig. 6.6). Fig. 6.6 shows a significant number of perpendicular lines found in both the upward and downward parts of the trajectory as compared to the number of parallel lines. As expected, there are a greater number of perpendicular lines found when travelling downwards as there are a greater number of stairs visible in the frame for typical texting use case device angles. Thresholding the plot of Fig. 6.6, we have an indication of when the device is travelling on a staircase (see Fig. 6.7).

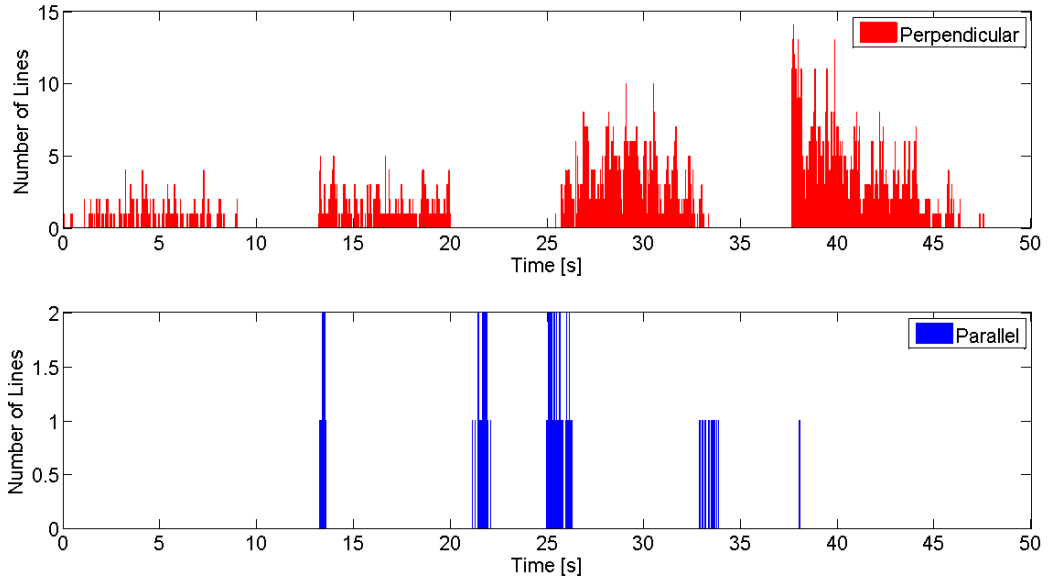


Fig. 6.5. Total number of lines found perpendicular or parallel to the device's direction of motion for the stairs texting context with the device held in a landscape orientation.

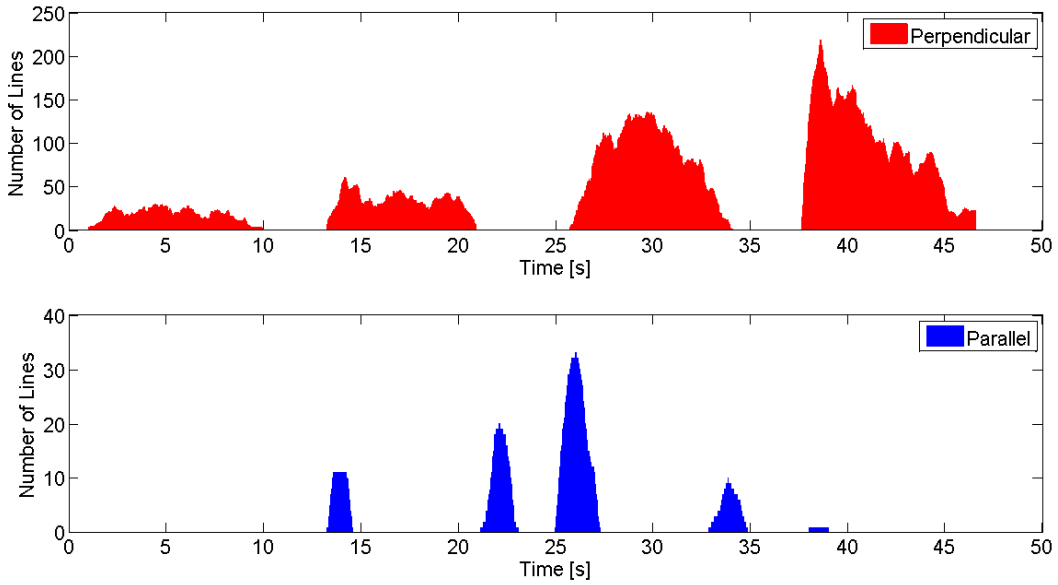


Fig. 6.6. Total number of lines found perpendicular or parallel to the device's direction of motion, integrated over a moving window of 30 frames (1 second), for the stairs texting context with the device held in a landscape orientation.

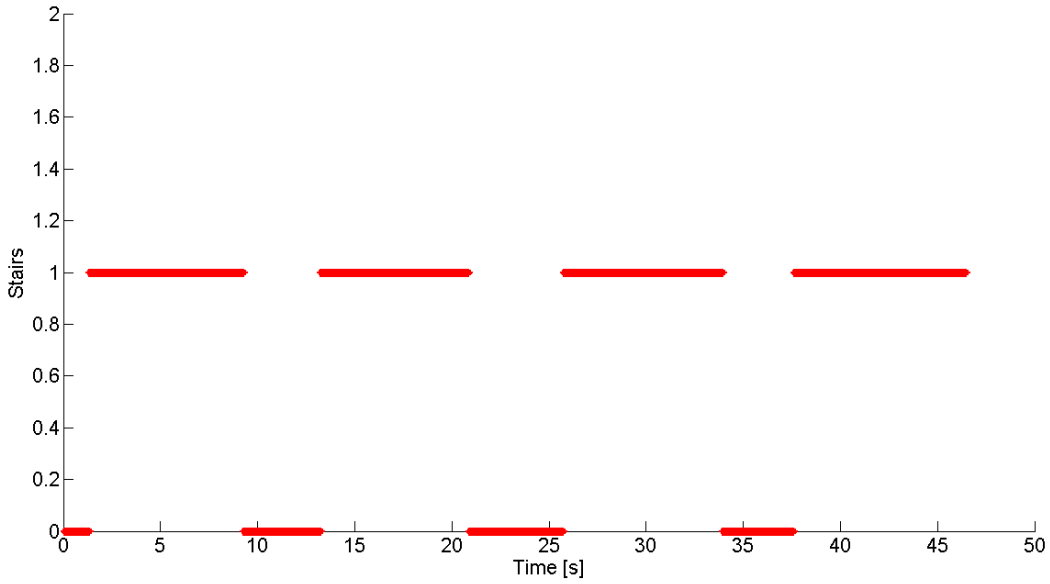


Fig. 6.7. Stairs mode indicator for the stairs texting context with the device held in a landscape orientation.

6.3 Texting Use Case – Tiled Floor

An important aspect of reliably detecting the stairs texting use case is the ability to differentiate between motion on stairs (which consists of predominantly perpendicular lines in the frame) and motion on a single floor where the ground has a repeating line-based pattern (e.g. tiles, linoleum, etc.). The solution is to look at the number of parallel lines in the frame in addition to the perpendicular lines. On a staircase, the number of perpendicular lines should far outweigh any momentary parallel lines found. Data collected on a tile floor, however, shows that both types of lines are present in the frame (see Fig. 6.8). By checking that we do not have a strong parallel component, we can distinguish motion on real stairs from motion on patterned ground. Failing to do so would result in a determination of the stairs texting use case for the majority of the trajectory as shown in Fig 6.9.

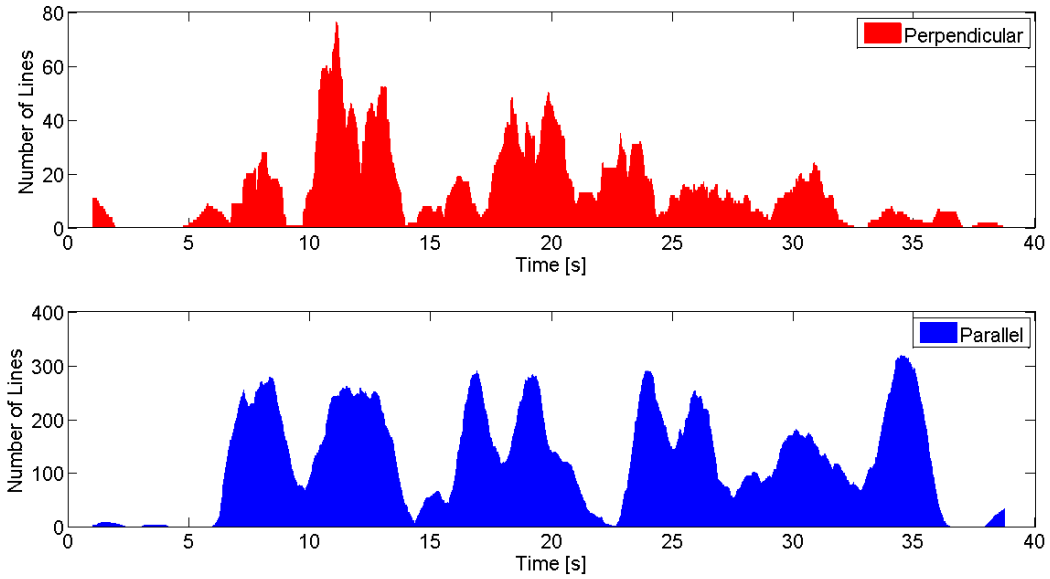


Fig. 6.8. Total number of lines found perpendicular or parallel to the device's direction of motion, integrated over a moving window of 30 frames (1 second) for the single floor texting context on a tiled floor with the device held in a landscape orientation.

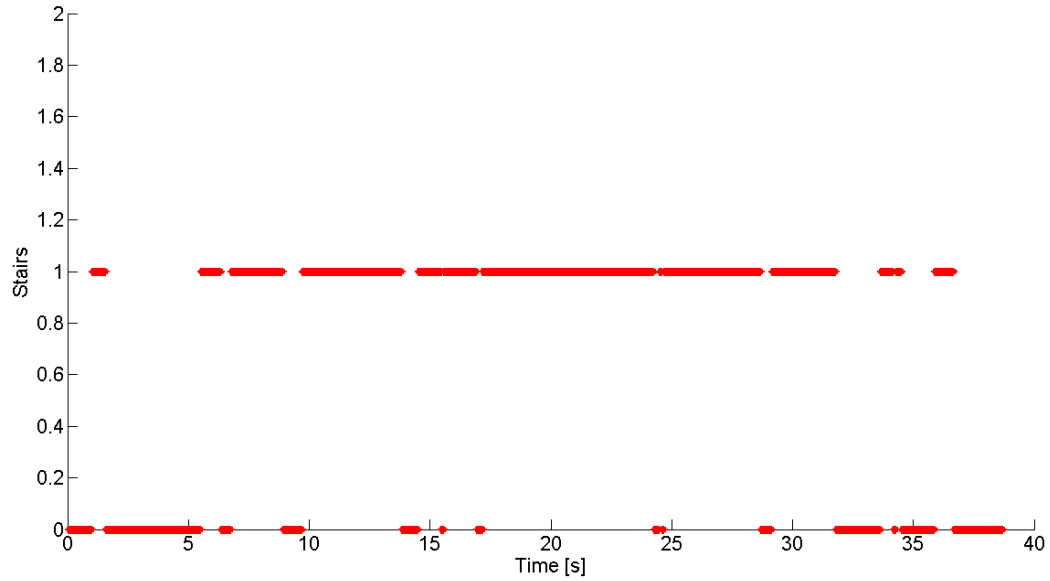


Fig. 6.9. Stairs indicator incorrectly reporting the stairs context.

6.4 Calling Use Case – Single Floor vs. Stairs

The single floor calling use case can be differentiated from the stairs calling use case by examining the ratio of the leveled horizontal and vertical components of the aggregated flow after integration over a small window. Examining Fig. 6.10, it can be seen that the flow ratio is largely below a value of 0.5 and centered around 0.2-0.3 when traversing a single floor.

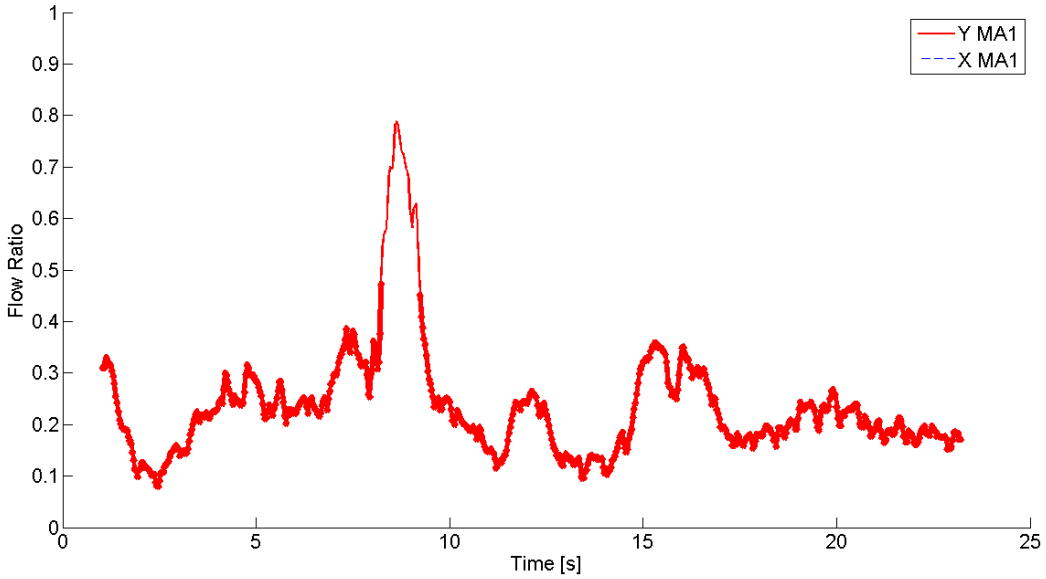


Fig. 6.10. Ratio of integrated Y to X flow over a moving window of 30 frames (1 second) for the single floor calling context with the device held in a landscape orientation.

The dataset collected in the stairs calling context involved walking down then back up two flights of stairs. This was repeated twice in each dataset. In between each flight of stairs was a small portion of level ground where the user of the device performed a 180 degree heading change in the form of a small U-turn around the center railing/bannister. As expected, the values of the integrated flow ratios previously shown for the single floor calling context become more interesting when compared to those obtained in the stairs calling context. When the device is

undergoing motion on stairs, the flow ratio exceeds the 0.5 threshold, only dropping below 0.5 during the times when the user is undergoing a heading change whilst on the same floor. Thresholding the flow ratios of Fig. 6.11, we obtain a simple metric for determining when the device is on stairs. The results of the stairs classification can be found in Fig. 6.12. After determining that the device is undergoing motion on stairs, we must determine whether the user is travelling up or down the stairs. Knowing that the user initially travelled down, there can be seen a direct correlation between the direction of motion and the sign of the y-component of the aggregated optical flow provided in Fig. 6.13. That is, a positive y-component indicates downwards motion (and vice versa). The determination of direction using this scheme can be found in Fig 6.14. Coupled with the stairs classification of Fig. 6.12, there is now a clear indication of the device's context.

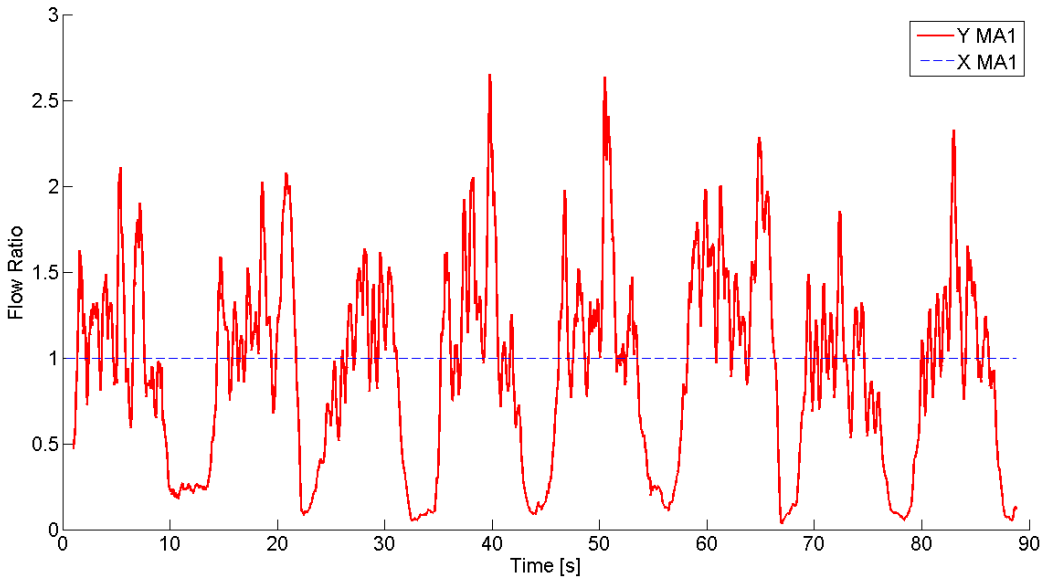


Fig. 6.11. Ratio of integrated Y to X flow over a moving window of 30 frames (1 second) for the stairs calling context with the device held in a reverse landscape orientation.

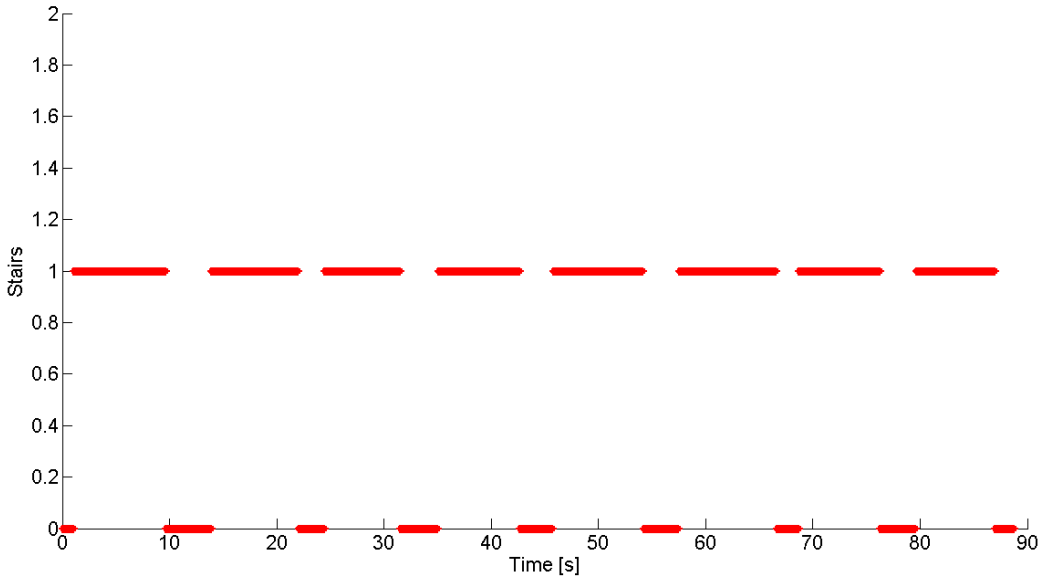


Fig. 6.12. Stairs context indicator for the stairs calling context with the device held in a reverse landscape orientation.

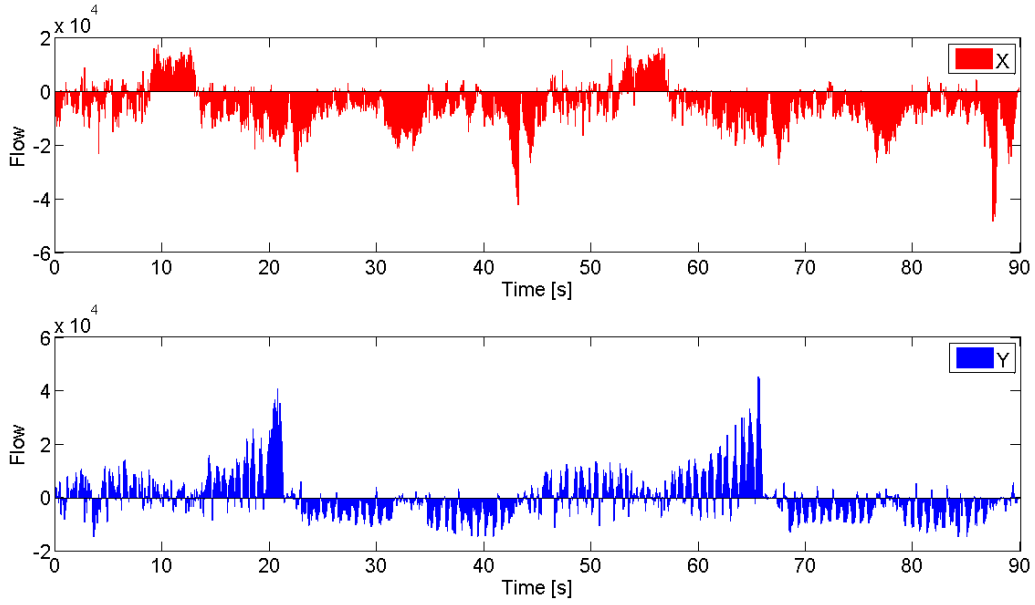


Fig. 6.13. X- and y-components of the aggregated optical flow for the stairs calling context with the device held in a reverse landscape orientation.

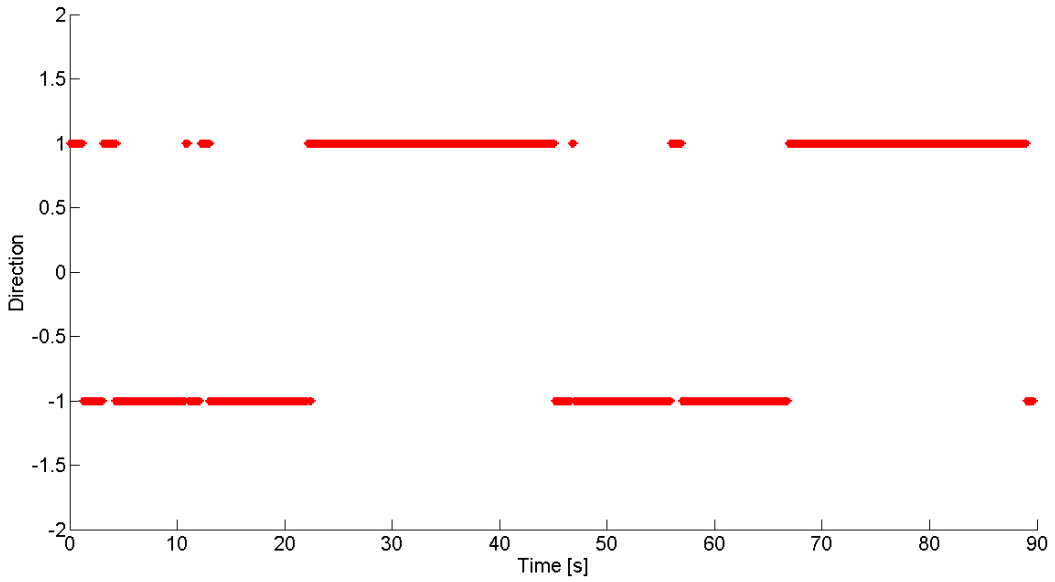


Fig. 6.14. Up/down direction determination using the positive/negative sign of the optical flow. A value of +1 indicates upwards motion while a value of -1 indicates downwards motion.

Chapter 7: Results for INS/Vision Integration

The device used for image and inertial data acquisition was the same Samsung Galaxy S3 used in the previous two chapters. The camera on the back of the phone (i.e. facing away from the screen) was used to capture video in specific scenarios while inertial data was collected in the background. The video files representing the various test cases were then processed offline using the algorithm described in Chapters 3 and 4 and used to generate the context and misalignment updates to be supplied to an offline PDR emulator. The performance of the vision aiding module is demonstrated by comparing the performance of the PDR-based navigation solution using user context and device misalignment updates from the vision aiding module with the navigation solution obtained without the use of these updates [56, 57].

The vision aiding module is assessed for its impact on PDR performance in seven common use cases: (1) fidgeting the phone throughout a trajectory on a single floor ("fidgeting"), (2) navigating indoors with the camera facing the ground at a constant misalignment ("single floor texting" with constant misalignment), (3) navigating indoors with the camera facing the ground while undergoing changes in misalignment ("single floor texting" with changing misalignments), (4) navigating indoors with changing misalignments while the phone is held vertical ("vertical"), (5) navigating indoors while the phone is dangling and changing misalignments ("dangling"), (6) traversing stairs with the camera facing the ground ("stairs texting") and (7) traversing stairs with the camera on ear ("stairs calling"). For each of these use cases, the context will be determined and the relevant information will be used by a PDR-based navigation scheme to constrain the solution accordingly by applying the appropriate updates to the internal filter. For example, when the phone is detected as being in the fidgeting or single floor calling use case, the height will be forced to remain constant even though the sensor drift is causing it to vary. When on stairs, the

time spent in the context is combined with directional information to obtain an indication of whether a positive or negative floor change has occurred. In contexts where motion is constrained to a single floor, the addition of vision-based misalignment estimation is assessed for its impact on the solution.

These external updates will improve the internally computed values of the respective quantities and in some cases, will provide missing information. For example, some of the currently available smartphones and tablets do not contain a barometer or pressure sensor. In the absence of a barometer, a PDR-based navigation system has no observability on the height when indoors and will be unable to detect floor changes. In this situation, the vision aiding module will be the only means of providing the missing height information. Furthermore, PDR-based navigation schemes that do not take into account changing device misalignments can quickly lead to an unusable solution. Though misalignment estimation can be performed based on the forces measured by the inertial sensors, there are cases in which users with very slow walking speeds or problematic gaits can cause sensor-based methods to fail or perform sub-optimally. In these cases, vision-based methods can serve to reduce the number of failure cases. A description of each test and the corresponding results are as follows [56].

7.1 Texting Use Case

7.1.1 Texting Use Case – Fidgeting

The first texting use case trajectory took the form of a user walking in one rectangular loop inside of an office building at a slow speed. The phone was held in the texting use case at zero misalignment for the full duration of the test. The purpose of this test was to evaluate the performance of the vision aiding module in the presence of static and/or fidgeting periods when

the user is walking slowly. Upon reaching each corner of the loop, the user was asked to remain static or fidget with the phone as if they were reading an article or typing a message, respectively. The duration of the test was approximately 2.5 minutes. Fig. 7.1.1 shows the misalignment estimation provided by the vision aiding module. The context classification with vision aiding is shown in Fig. 7.1.2. It should be noted that there is an erratic misalignment value reported in the first approximately 15 seconds of the trajectory. This effect is also seen throughout all subsequent trajectories not due to the vision aiding module but rather the experimental design. In each trajectory, the user began logging inertial data on the phone for approximately 10-15 seconds before starting the video recording application. As such, the start of the video data is offset with respect to the start of the inertial data by the observed delay. The correct vision-based values follow thereafter.

Examining Fig. 7.1.1, it can be seen that the vision-derived device misalignment is approximately zero degrees except in a few regions where the device angle rises to 90 degrees. It Comparing Fig. 7.1.1 with 7.1.2, we see that these regions coincide with the regions in which the vision module reports fidgeting behavior. As such, the incorrect misalignments reported in these regions are ignored and do not affect the positioning solution.

The effect of integrating the vision-derived misalignment and context on the PDR-based trajectory can be found in Fig. 7.1.3 and Fig. 7.1.4. Fig. 7.1.3 (a) shows the trajectory obtained using a fixed zero degree misalignment value for the full duration of the trajectory. In this case, the addition of vision-based misalignment values which are also near zero makes a negligible impact on the position solution, as shown in Fig. 7.1.3 (b).

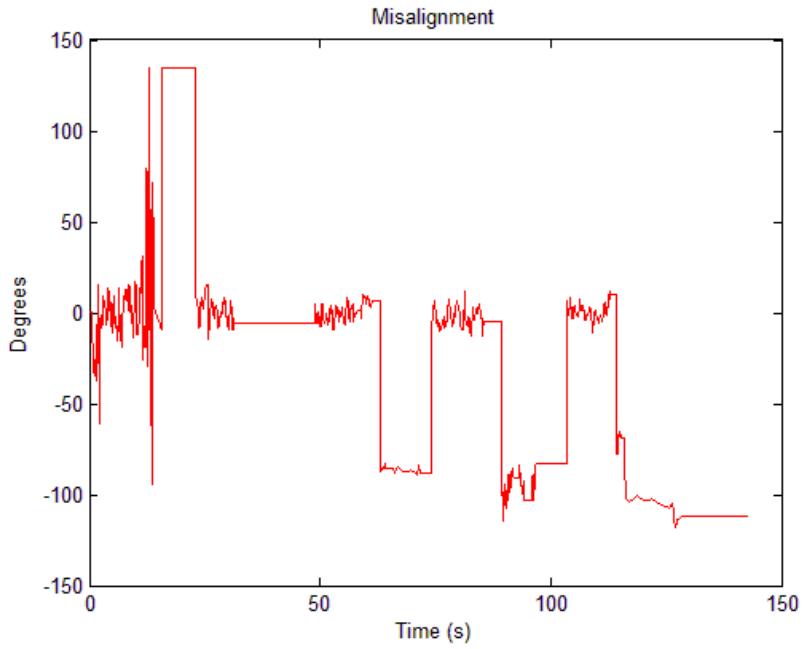


Fig. 7.1.1. Device misalignment estimation obtained with vision aiding.

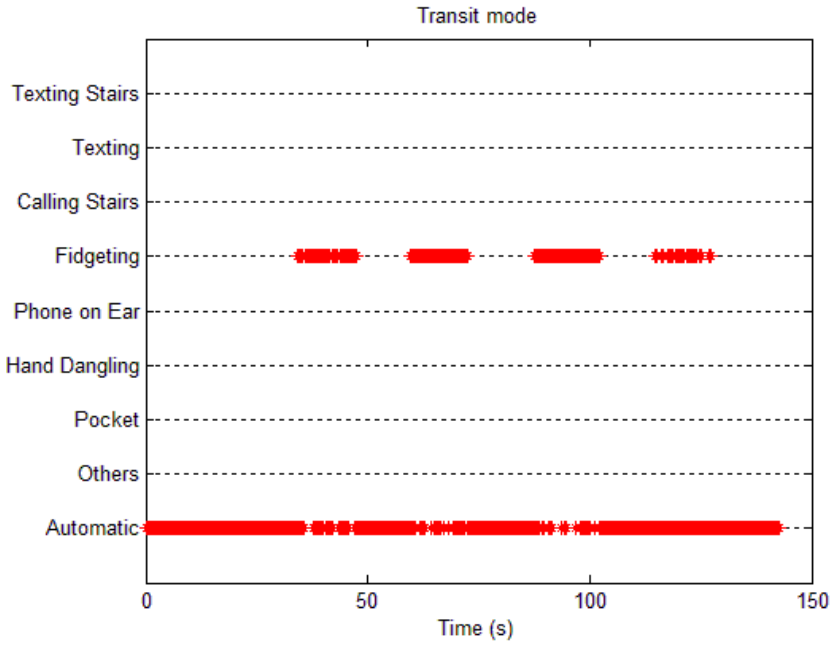
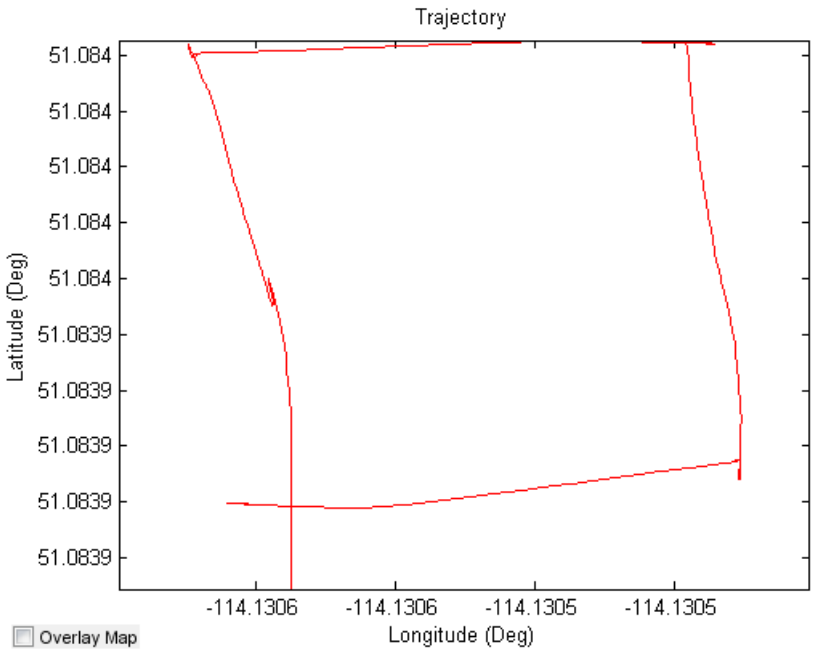
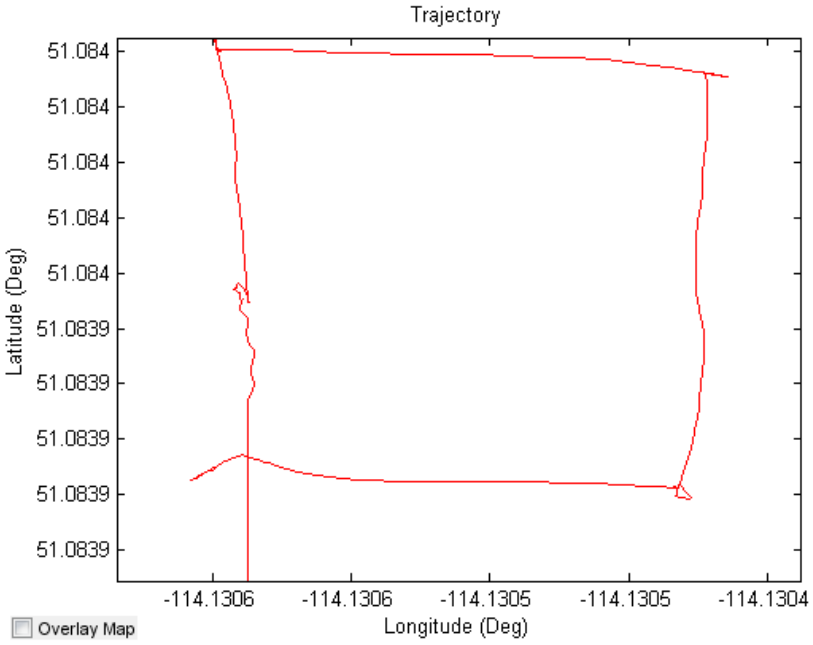


Fig. 7.1.2. Context classification obtained with vision aiding.

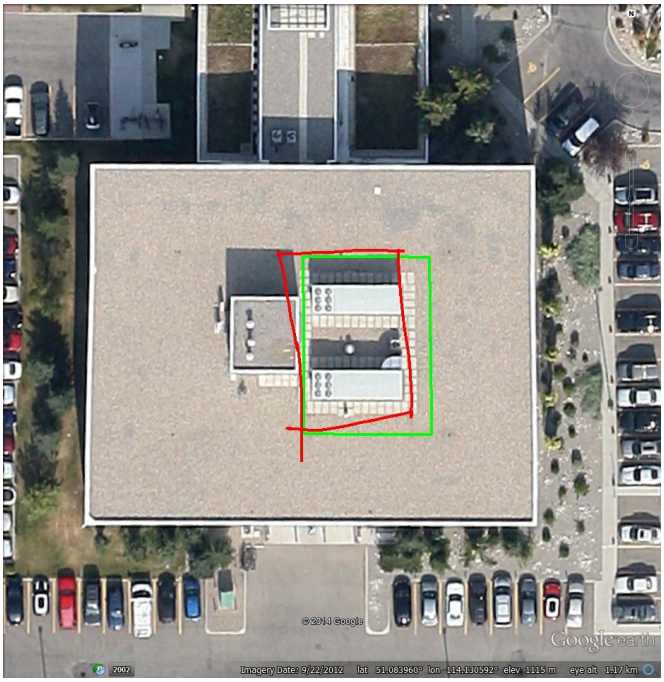


(a)



(b)

Fig. 7.1.3. Trajectory obtained (a) using a fixed zero degree misalignment value and (b) using vision-based misalignment estimation.



(a)

(b)

Fig. 7.1.4. Real-world trajectory obtained (a) using a fixed zero degree misalignment value and (b) using vision-based misalignment estimation. The reference path is shown in green.

7.1.2 Texting Use Case – Constant Misalignment

The second texting use case trajectory took the form of a user walking four loops inside of an office building at a slow speed. Just as in the previous test, the phone was held in the texting use case at zero misalignment for the full duration of the test. The purpose of this test was to assess the performance of vision-based misalignment estimation for slow walking speeds on a longer time scale void of any static or fidgeting periods (i.e. no zero velocity updates or position constraints). The duration of the test was approximately 3.5 minutes.

Fig 7.2.1 shows the misalignment estimation provided by the vision aiding module. The trajectories obtained using a fixed misalignment value of zero degrees as well as using the misalignment estimation from the vision aiding module are shown in Fig 7.2.2 (a) and (b), respectively. The same respective real-world trajectories can be found in Fig. 7.2.3 (a) and (b).

The vision-based misalignment values of Fig. 7.2.1 show strong consistency in the long term with no time-dependent bias or offset. Incorporating the misalignment values from the vision module into the PDR solution has the effect of maintaining the upright and rectangular shape of the trajectory throughout all four loops, bringing the trajectory of Fig. 7.2.2 (b) closer in resemblance to the fixed misalignment trajectory of Fig. 7.2.2 (a).

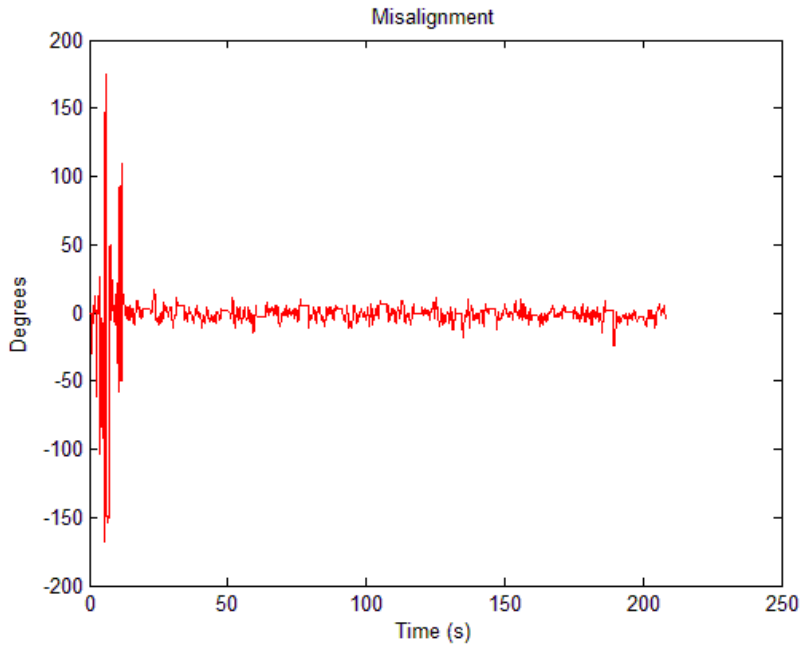
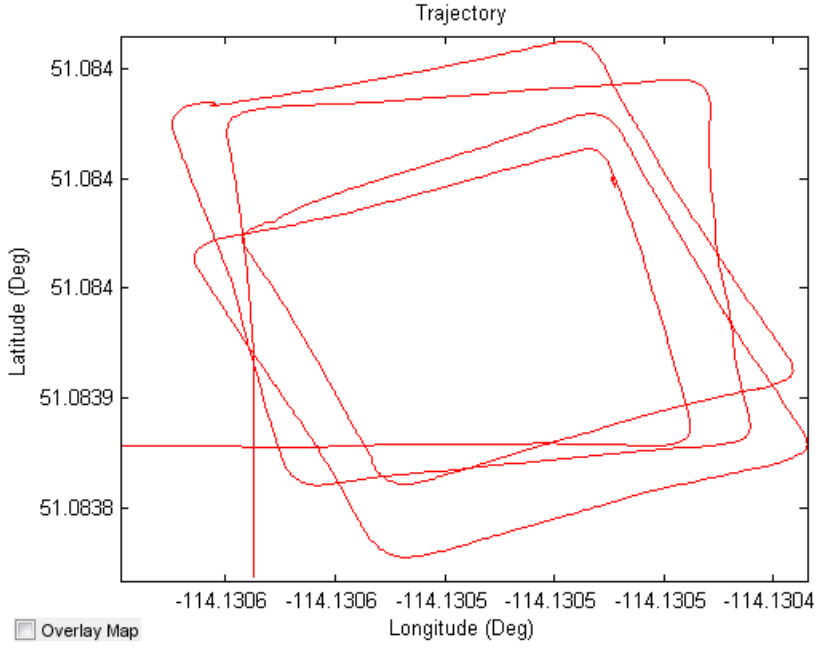
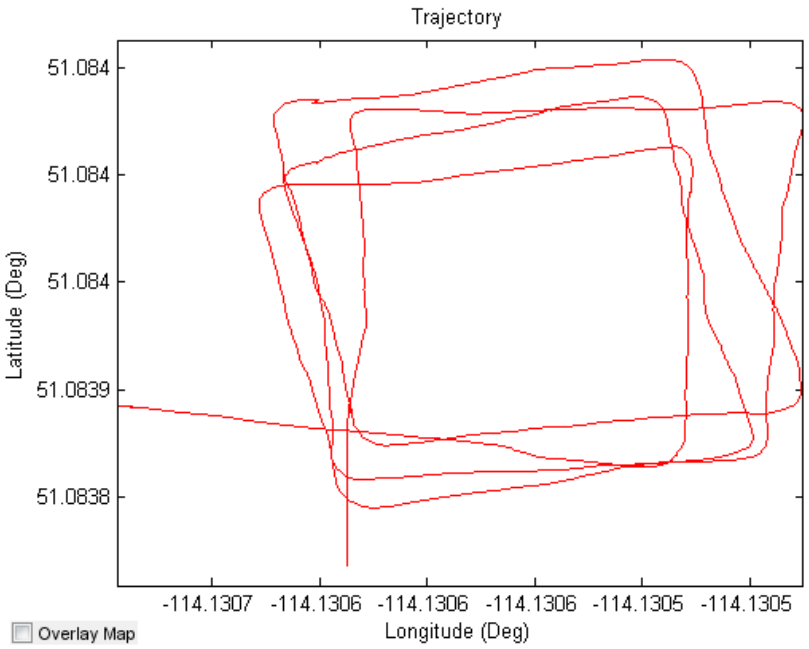


Fig. 7.2.1. Device misalignment estimation obtained with vision aiding.

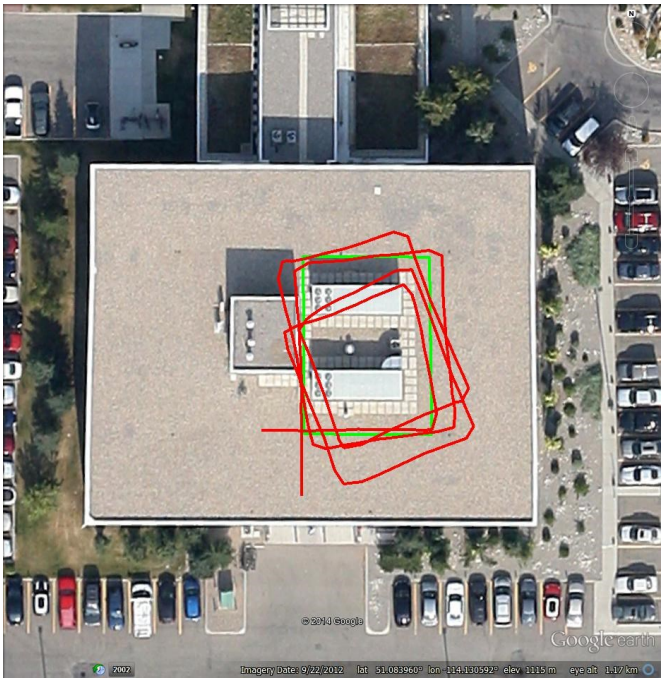


(a)

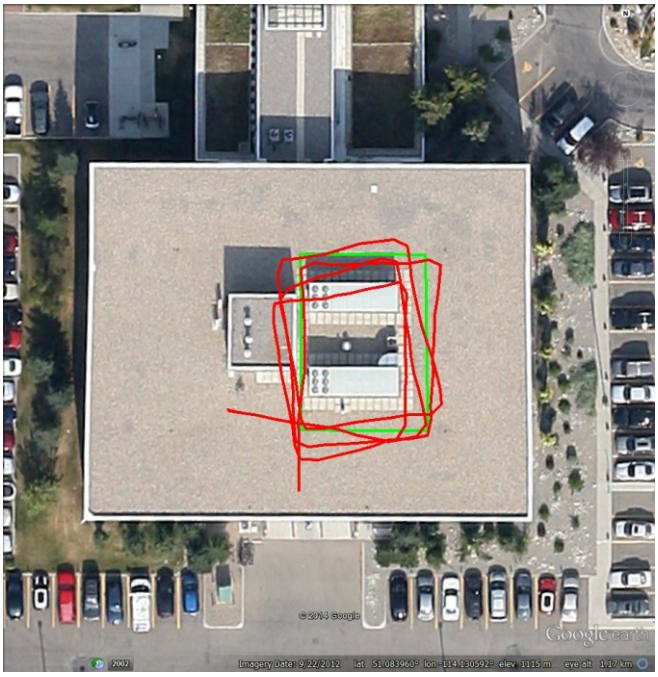


(b)

Fig. 7.2.2. Trajectory obtained (a) without vision aiding and (b) with vision aiding.



(a)



(b)

Fig. 7.2.3. Real-world trajectory obtained (a) using a fixed zero degree misalignment value and (b) using vision-based misalignment estimation. The reference path is shown in green.

7.1.3 Texting Use Case – Changing Misalignment

The third texting use case trajectory took the form of a user walking the same loops as that of the previous test. In contrast to the previous trajectories, the purpose of this test was to assess the performance of vision-based misalignment estimation for a device with changing misalignments and a slow walking speed. To this end, the user was asked to walk three loops and change the phone orientation after the completion of each loop. The duration of the test was approximately 3 minutes.

Fig. 7.3.1 shows the misalignment values provided by the vision aiding module. The trajectories obtained using a fixed zero degree misalignment value and using misalignment estimation from the vision aiding module are shown in Fig. 7.3.2 (a) and (b), respectively.

Examining Fig. 7.3.1, it is easy to deduce that the device was held with a misalignment of zero degrees in the first loop, -90 degrees in the second loop, and +90 degrees in the third. The vision-based misalignment values in Fig. 7.3.1 provide a clear picture of the device angle in each section of the trajectory. Furthermore, the vision-based values respond quickly to a change in misalignment with sharp transition points between the different orientations. Incorporating the vision-derived values into the PDR navigation solution of Fig. 7.3.2 (a) shows a dramatic improvement in the trajectory, shown in Fig. 7.3.2 (b). The maximum observed position error was approximately 20 meters without vision aiding, dropping to approximately 2 meters using vision-based misalignment estimation.

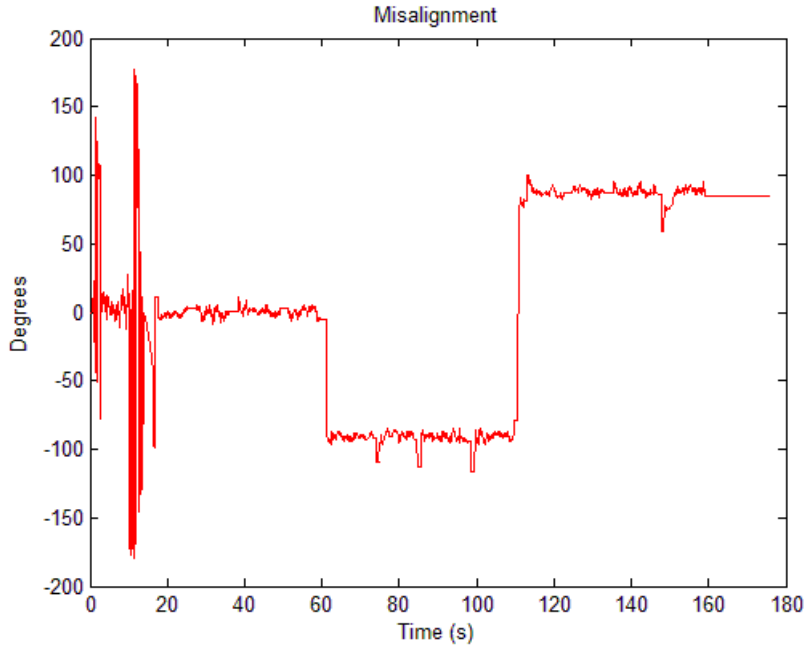
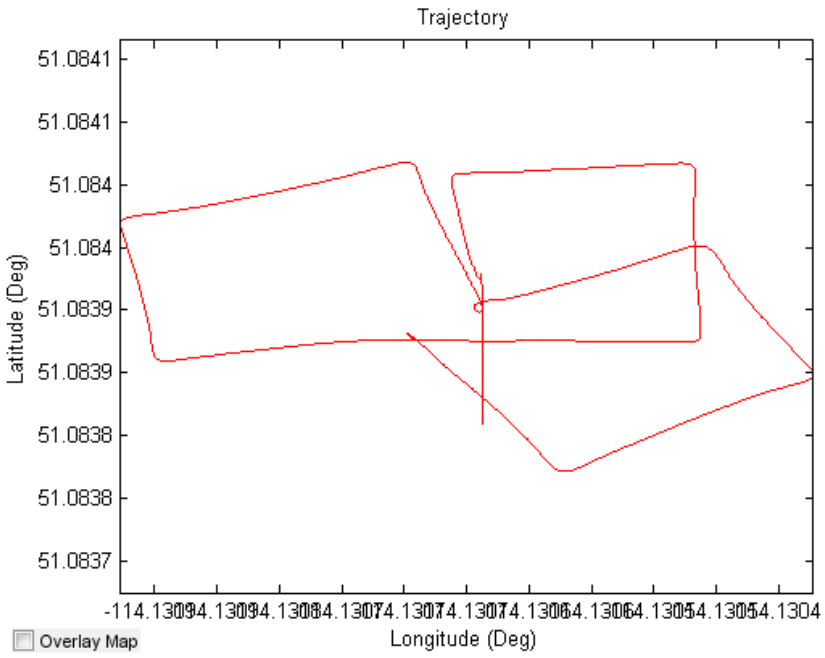
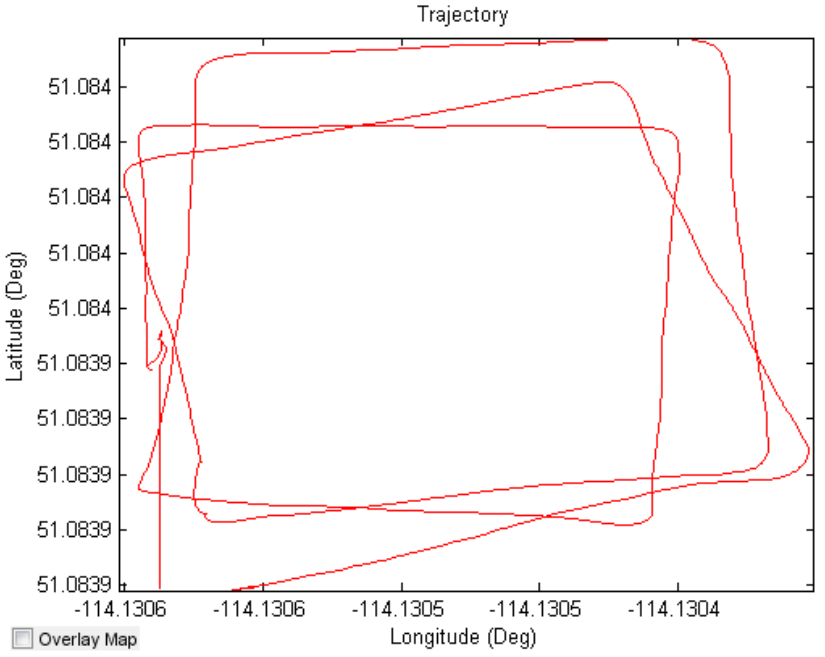


Fig. 7.3.1. Device misalignment estimation obtained with vision aiding.

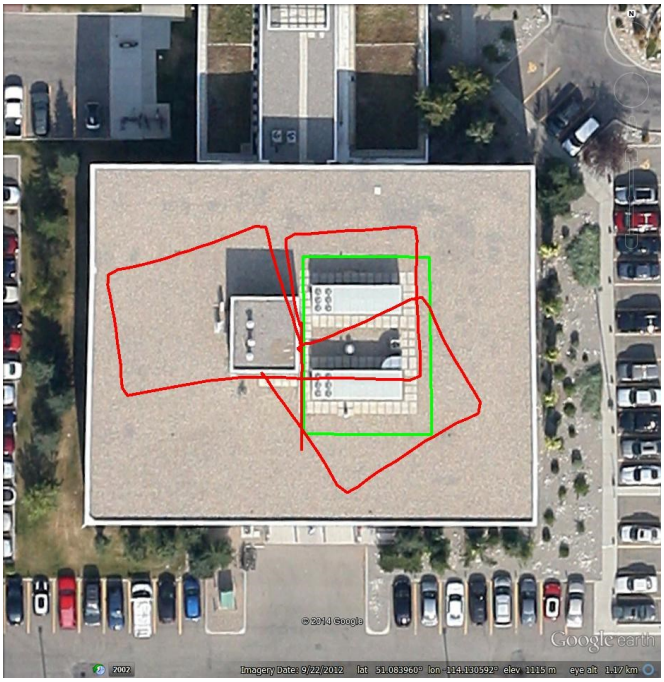


(a)



(b)

Fig. 7.3.2. Trajectory obtained (a) without vision aiding and (b) with vision aiding.



(a)

(b)

Fig. 7.3.3. Real-world trajectory obtained (a) using a fixed zero degree misalignment value and (b) using vision-based misalignment estimation. The reference path is shown in green.

7.2 Vertical Use Case

The vertical use case trajectory took the form of a user walking two loops in an indoor parkade. The purpose of this test was to assess the performance of vision-based misalignment for a trajectory with changing misalignments while the device was held vertically. The user was asked to change the phone orientation at random points in the trajectory. The duration of the test was approximately 4.5 minutes. Fig. 7.4.1 shows the misalignment estimation provided by the vision aiding module. The trajectories obtained using a fixed zero degree misalignment value and using misalignment estimation from the vision aiding module are shown in Fig. 7.4.2 (a) and (b), respectively.

Similar to the previous test case, Fig. 7.4.2 (a) shows that using a fixed zero degree misalignment value in the vertical use case also fails to produce a useful navigation solution in the presence of changing misalignments. Fig. 7.4.1 provides a clear picture of the device orientation in different sections of the trajectory and incorporating these misalignment values into the PDR solution yields a significantly more accurate result, as shown in Fig. 7.4.2 (b). The maximum observed position error was approximately 60 meters without vision aiding, dropping to approximately 8 meters using vision-based misalignment estimation.

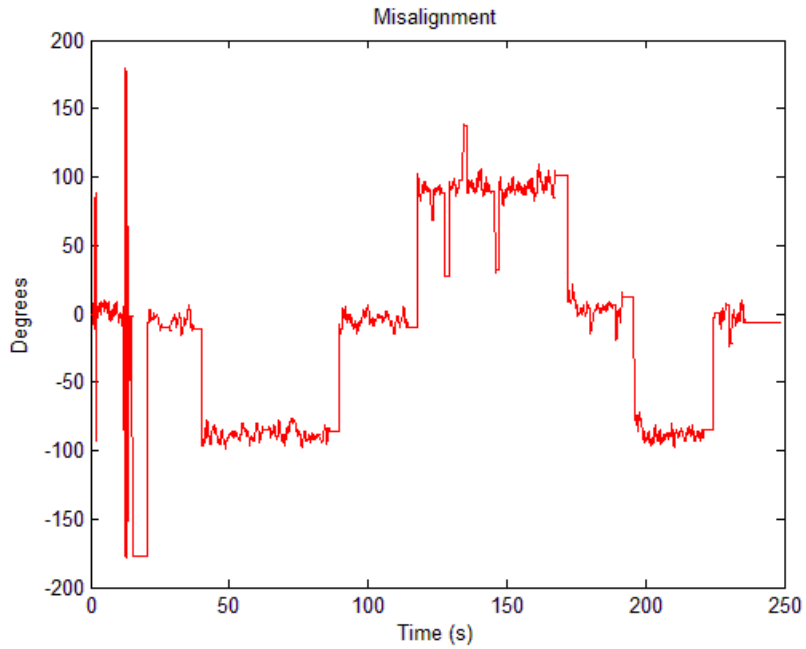
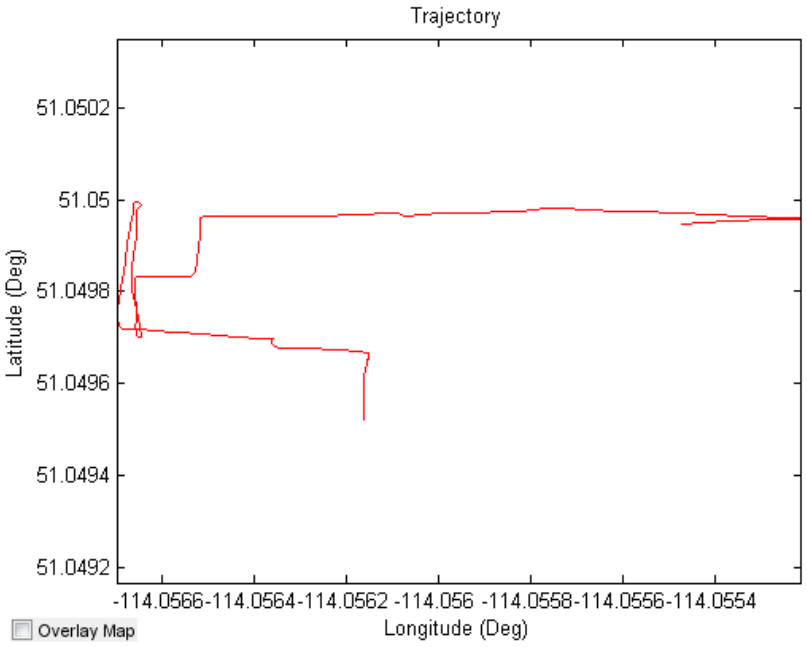
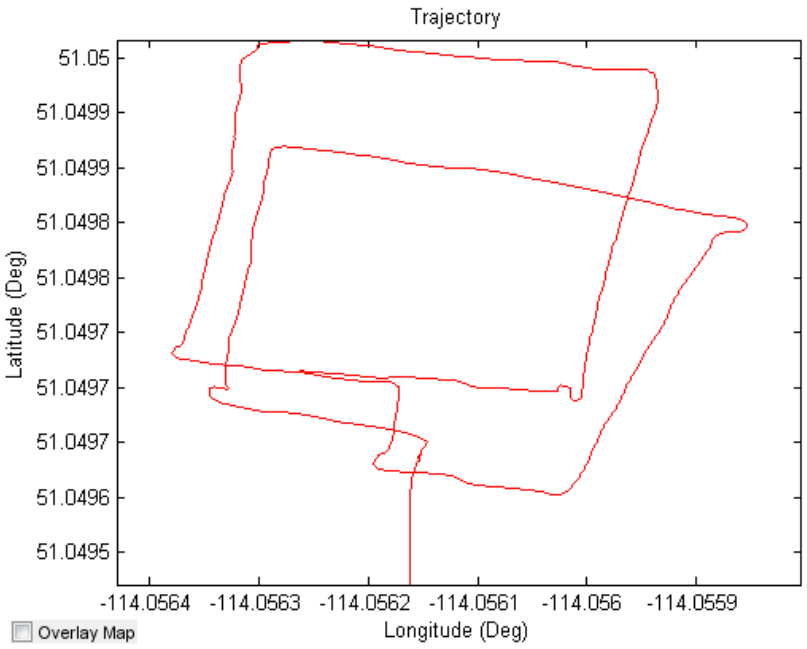


Fig. 7.4.1. Device misalignment estimation obtained with vision aiding.

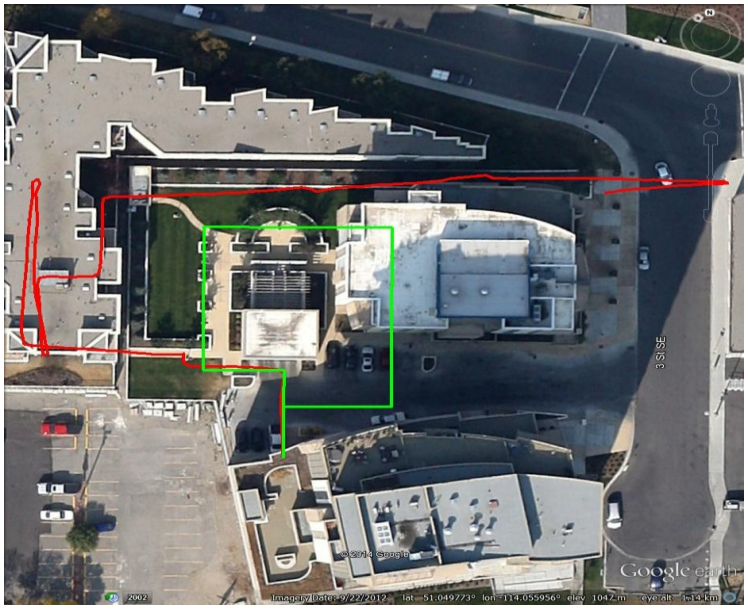


(a)

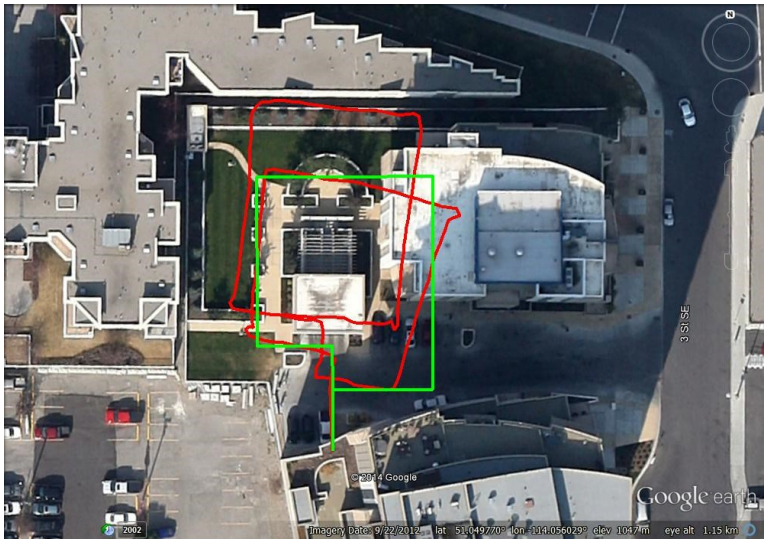


(b)

Fig. 7.4.2. Trajectory obtained (a) without vision aiding and (b) with vision aiding.



(a)



(b)

Fig. 7.4.3. Real-world trajectory obtained (a) using a fixed zero degree misalignment value and (b) using vision-based misalignment estimation. The reference path is shown in green.

7.3 Dangling Use Case

The dangling use case trajectory took the form of a user walking down a hallway while holding the device in their hand in one of three orientations: 0, 180, and -90 degrees. Upon reaching the end of the hallway, the user was asked to transition to the next orientation in the list before walking back along the same path. The purpose of this test was to assess the performance of vision-based misalignment estimation for a trajectory with changing dangling misalignments. The duration of the test was just under 2 minutes. Fig. 7.5.1 shows the misalignment estimation provided by the vision aiding module. The trajectories obtained using a fixed zero degree misalignment value and using misalignment estimation from the vision aiding module are shown in Fig. 7.5.2 (a) and (b), respectively.

The improvements seen for the dangling tests echo those seen in the texting and vertical tests in which there were unpredictable and dramatic changes in the device orientation. Fig. 7.5.1 shows

that the device was clearly in a zero degree orientation on the first pass through the hallway before transitioning to the 180 and -90 degree orientation. Fig. 7.5.2 (a) shows the consequence of not taking into account the change in device orientation in the dangling use case. Incorporating the misalignment changes into the PDR solution yields a significantly more accurate result, as shown in Fig. 7.5.2 (b). It is important to note, however, that the ending point of the trajectory fell slightly short of the end of the hallway where the user actually ended the test. This can be attributed to the two spikes observed in the misalignment plot of Fig. 7.5.1 at time of approximately 82 and 92 seconds. Fig. 7.5.1 incorrectly reports a temporary spike from -90 to +90 degree misalignment when the user made no such change to the device. These two periods of incorrect misalignment estimation has the effect of projecting the user position 180 degrees backwards along the trajectory rather than forward in the direction of user motion. As such, the ending position of the PDR with vision aiding scheme falls short of the true ending position. The incorrect misalignment values reported in these regions may be attributed to a temporary disturbance to the optical flow measurements, such as the presence of other people momentarily walking alongside the user. The position error due to this anomaly is still negligible compared to the position error that would have been obtained using a fixed zero degree misalignment value. The maximum observed position error was approximately 80 meters without vision aiding, dropping to approximately 2 meters using vision-based misalignment estimation.

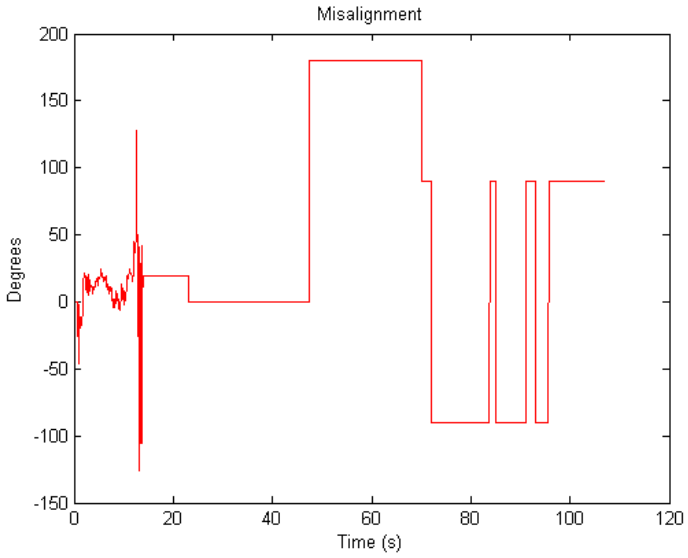
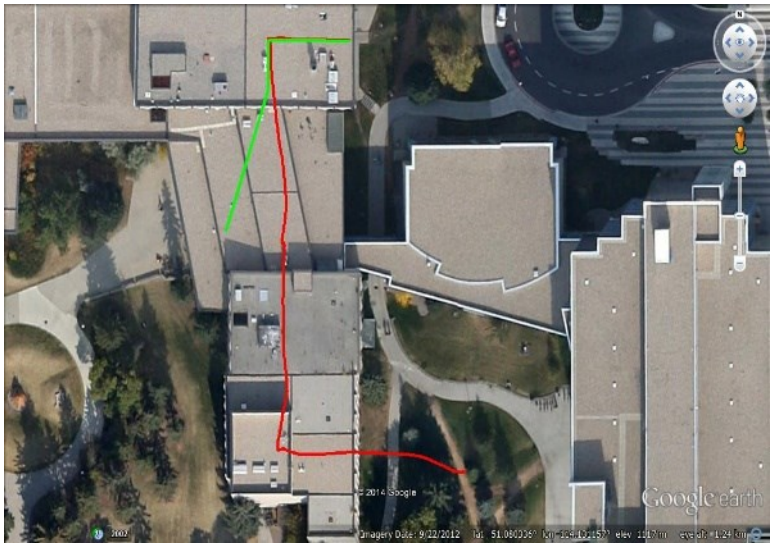
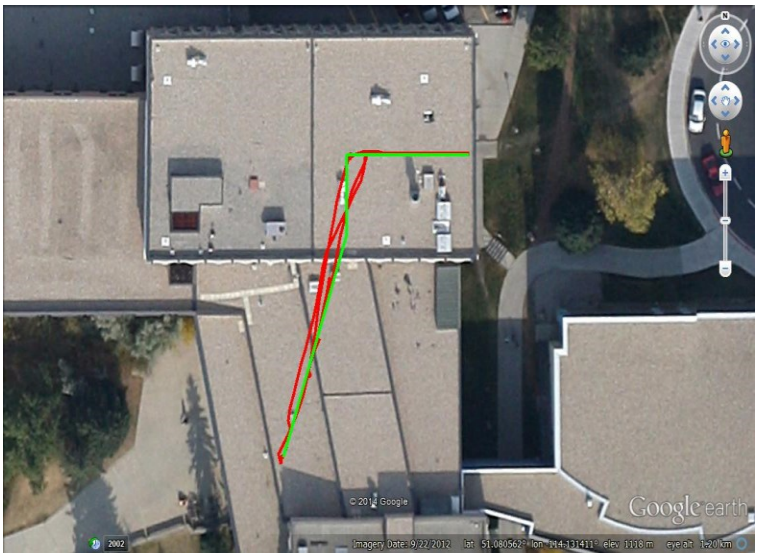


Fig. 7.5.1. Device misalignment estimation obtained with vision aiding.



(a)



(b)

Fig. 7.5.2. Real-world trajectory obtained (a) using a fixed zero degree misalignment value and (b) using vision-based misalignment estimation. The reference path is shown in green.

7.4 Multi-floor Texting Use Case

The multi-floor texting use case trajectory took the form of a user walking into the stairwell of an apartment building, traversing a flight of stairs up to the second floor and coming back down. This sequence was repeated three times for a total of six floor changes throughout the trajectory. The purpose of this test was to evaluate whether the device camera could successfully detect the stairs texting use case and, if so, when a floor change had occurred. The user was asked to keep the phone at a constant misalignment and the duration of the test was approximately 2 minutes.

Fig. 7.6.1 shows the number of lines detected in the images that were perpendicular and parallel to the direction of motion. The presence of lines perpendicular to the direction of motion was taken as being indicative of stairs, as shown in the stairs indicator of Fig. 7.6.2. As mentioned in Section 4.4, there are multiple methods in which the direction of the motion in the stairs texting use case

can be determined. In this implementation, the ratio of the leveled horizontal and vertical components of the aggregated optical flow after integration over a small moving window is chosen, as shown in Fig. 7.6.3, and thresholded to obtain an indication of the direction of travel as shown in Fig. 7.6.4.

The duration of time spent on each detected instance of stairs (from the stairs indicator of Fig. 7.6.2) is shown in Fig. 7.6.5 and used in conjunction with the direction indicator of Fig. 7.6.4 to obtain the floor change indicator of Fig. 7.6.6 under the assumption that a floor change has not occurred unless the user has spent a minimum of five seconds in any one detected instance of stairs. The floor change indicator can be used to update the current floor that the user is on, shown in Fig. 7.6.7, as well as the user context, shown in Fig. 7.6.8. Lastly, applying the detected context to the navigation solution yields the trajectory obtained in Fig. 7.6.9. The real-world trajectories corresponding to the cases without and with vision-based context detection can be found in Fig. 7.6.10 (a) and (b), respectively. Though the trajectories look identical, there is a very small difference in the general shape of the trajectories obtained. The subtle difference is that in the case of the trajectory obtained with vision aiding, the reduced stride length used when the device was determined to be on stairs led to a slightly more compact solution that is closer to the stairwell. The effect of the reduced stride length is evident as both trajectories were obtained using identical initial coordinates However, the primary advantage of using vision aiding in this scenario is the successful detection of floor changes.

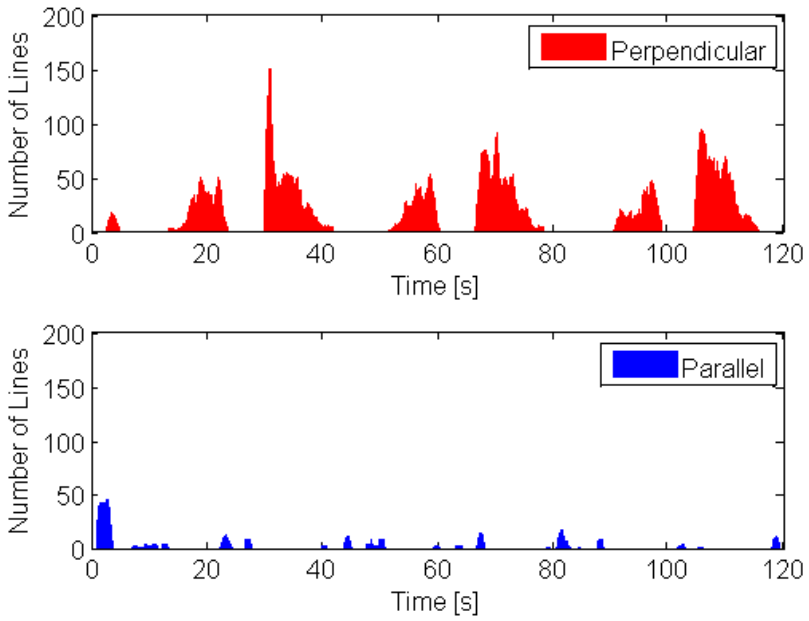


Fig. 7.6.1. Detection of perpendicular and parallel lines relative to the direction of motion.

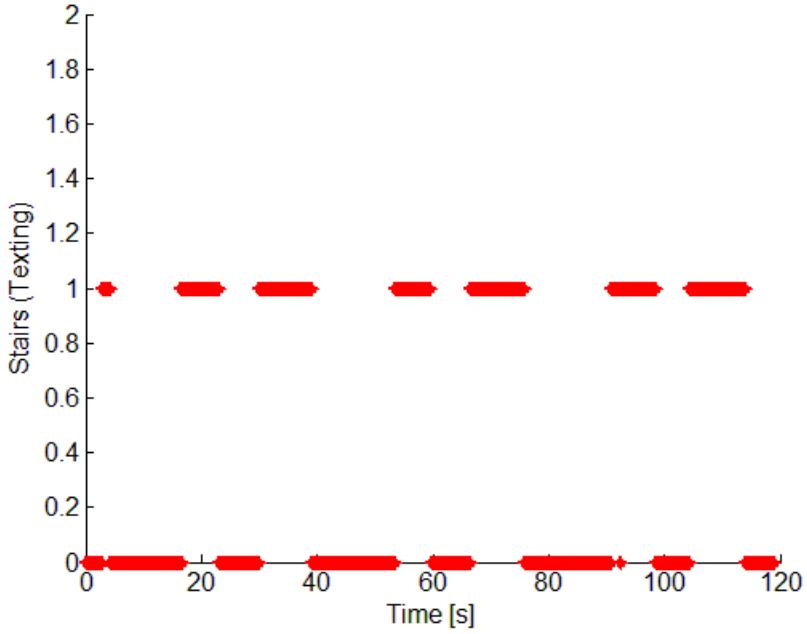


Fig. 7.6.2. Stairs detection indicator.

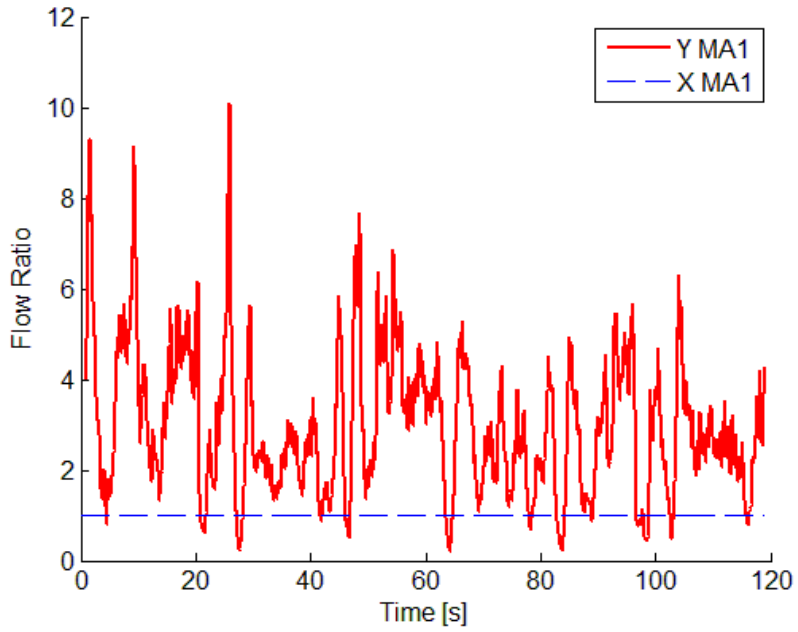


Fig. 7.6.3. Ratio of y- to x-flow integrated over a window of 30 frames (i.e. 1 second).

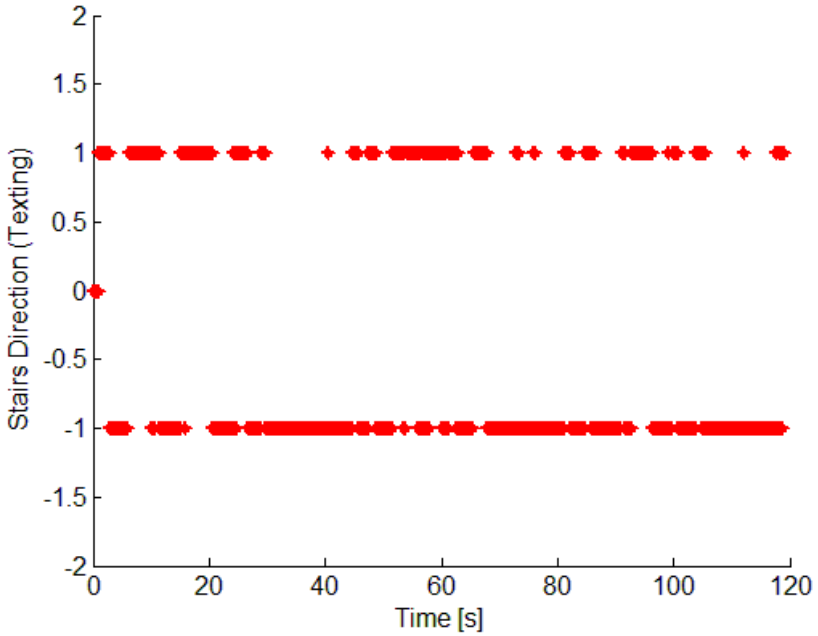


Fig. 7.6.4. Direction indicator.

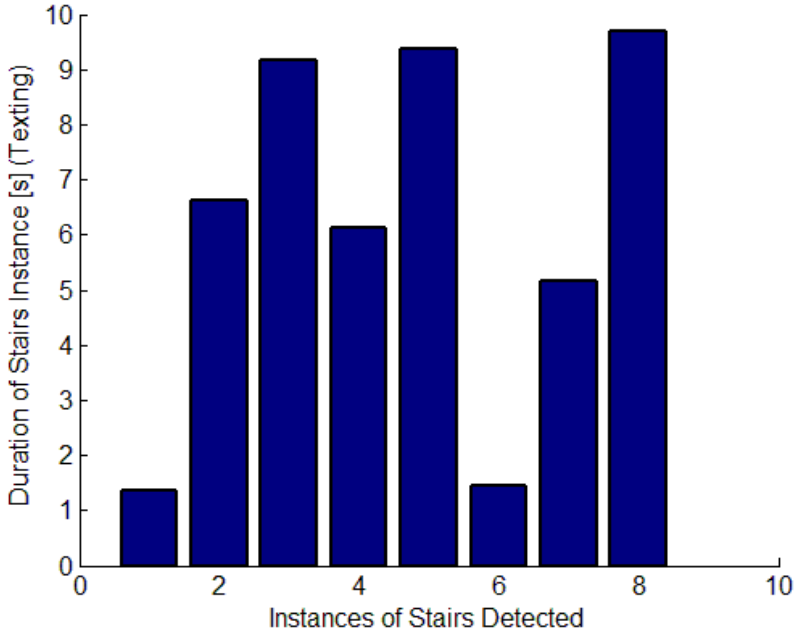


Fig. 7.6.5. Duration of detected instances of stairs.

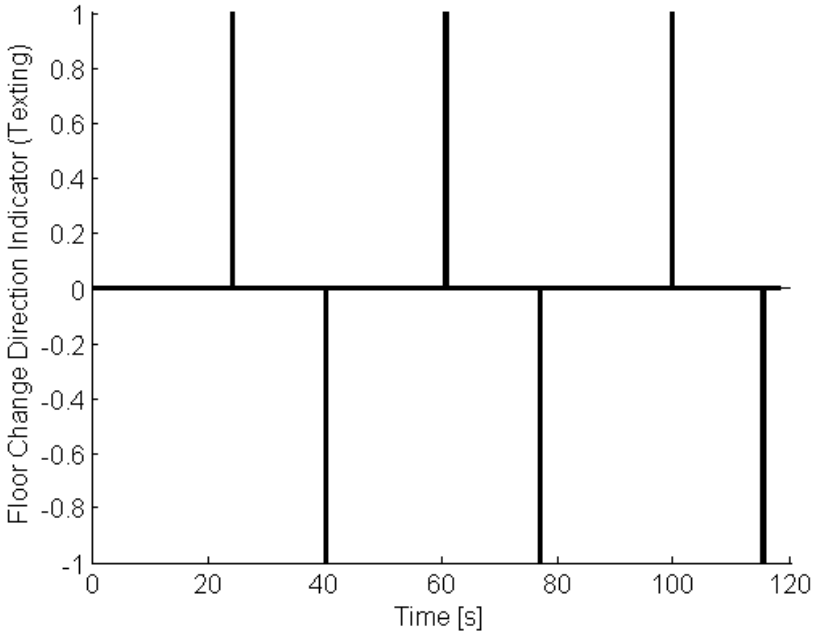


Fig. 7.6.6. Floor change indicator. A positive spike indicates a positive floor change while a negative spike indicates a negative floor change.

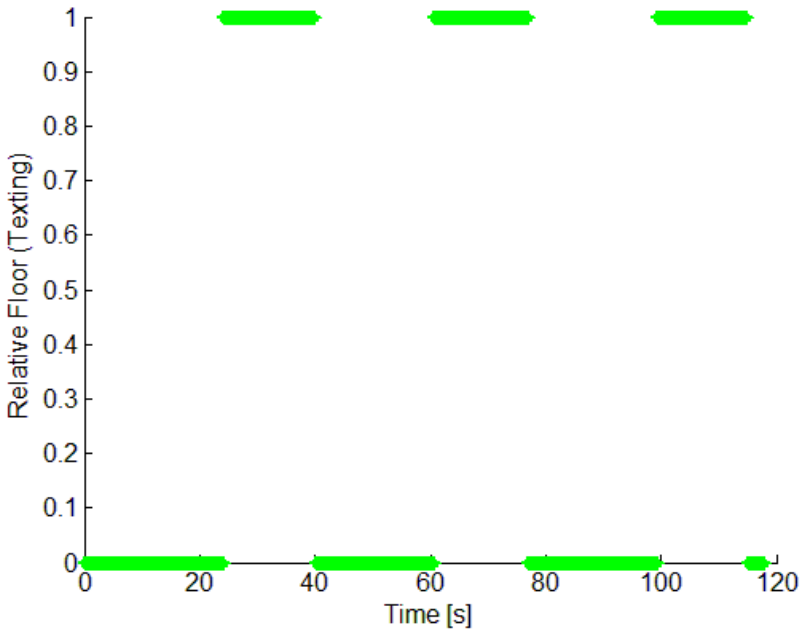


Fig. 7.6.7. Current floor.

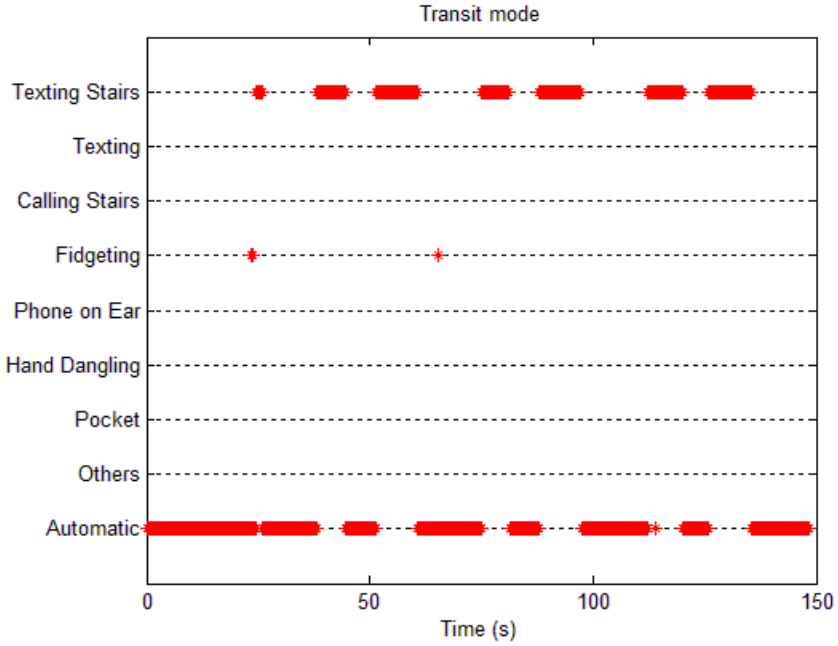
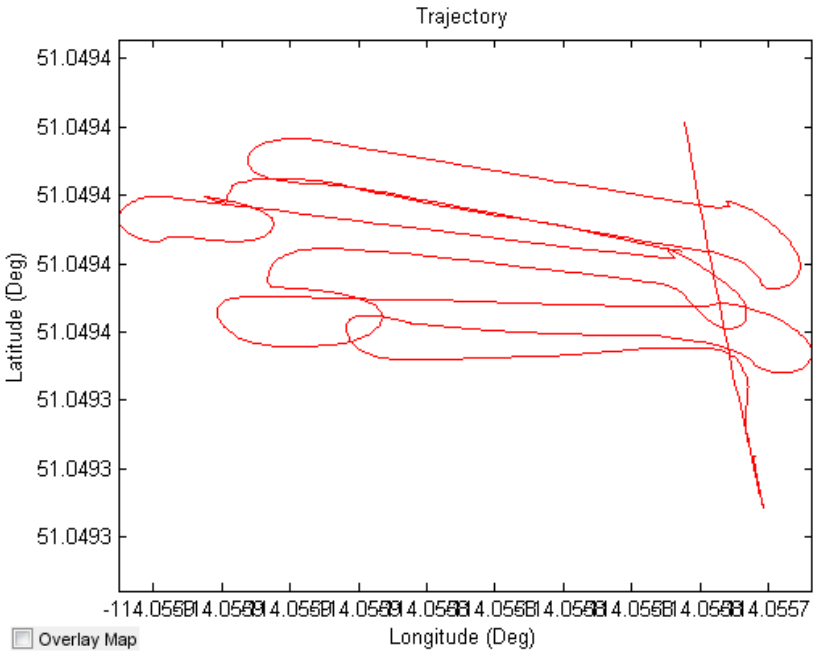
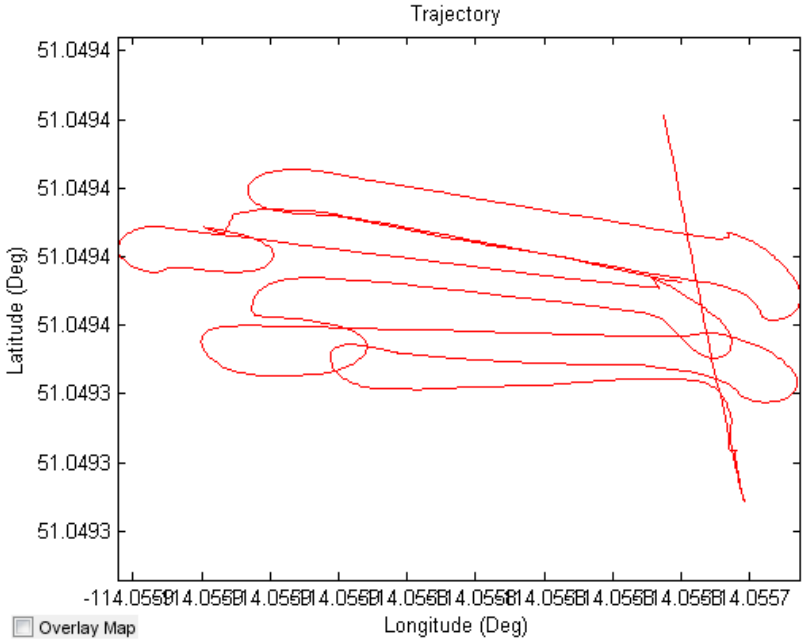


Fig. 7.6.8. Context classification obtained with vision aiding.

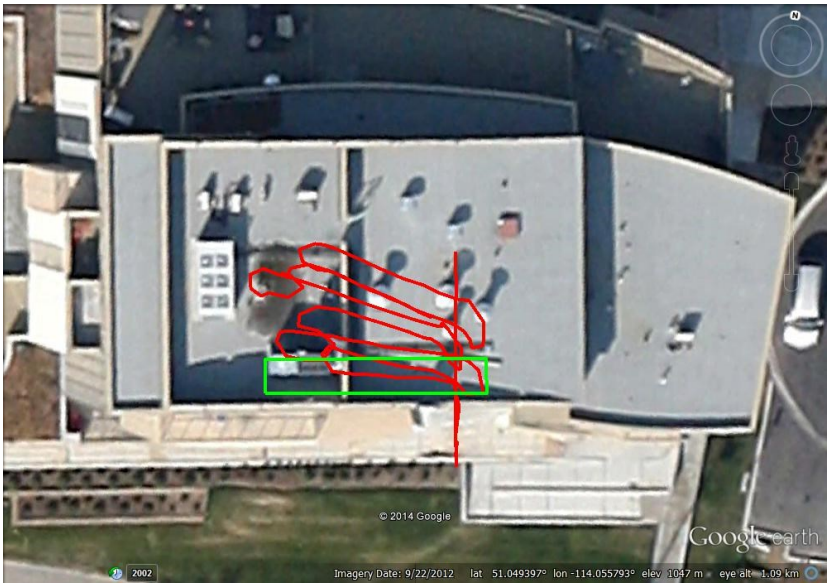


(a)

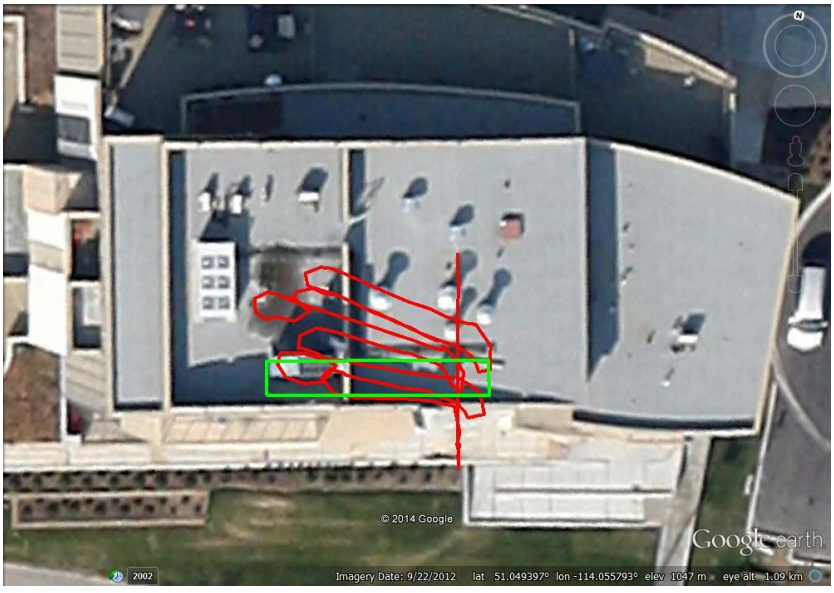


(b)

Fig 7.6.9. Trajectory obtained (a) without context detection and (b) using vision-based context detection.



(a)



(b)

Fig. 7.6.10. Real-world trajectories obtained (a) without context detection (b) using vision-based context detection. The reference path is shown in green.

7.5 Multi-floor Calling Use Case

The multi-floor calling use case trajectory was identical to that of the aforementioned texting use case trajectory. The user was asked to keep the phone at a constant misalignment as if they were on a call and the test took approximately 2 minutes.

Fig. 7.7.1 shows the ratio of the leveled horizontal and vertical components of the aggregated optical flow after integration over a small moving window. The regions of stairs are clearly identifiable as they exhibit flow ratios surrounding unity. The plot of Fig. 7.7.1 is thresholded and used to generate the stairs indicator plot of Fig. 7.7.2. The sign of the vertical component of the optical flow is used to form the direction indicator found in Fig. 7.7.3.

The duration of time spent on each detected instance of stairs (from the stairs indicator of Fig. 7.7.2) is shown in Fig. 7.7.4 and used in conjunction with the direction indicator of Fig. 7.7.3 to obtain the floor change indicator of Fig. 7.7.5. Just as in the stairs texting use case, it is assumed that a floor change has not occurred unless the user has spent a minimum of 5 seconds in any single detected instance of stairs. The floor change indicator is then used to update the current floor that the user is on, shown in Fig. 7.7.6, as well as the user context, shown in Fig. 7.7.7. Lastly, applying the detected context to the navigation solution yields the trajectory obtained in Fig. 7.7.8. The real-world trajectories obtained with and without vision-based context detection can be found in Fig. 7.7.9 (a) and (b), respectively. Similar to the stairs texting use case, there is a very small difference in the general shape of the trajectories obtained. However, the effect of the reduced stride length used in the stairs context has a more pronounced effect here as compared to the previous scenario as the solution is kept mostly within the boundaries of the stairwell. Furthermore, vision aiding again provides the successful detection of floor changes.

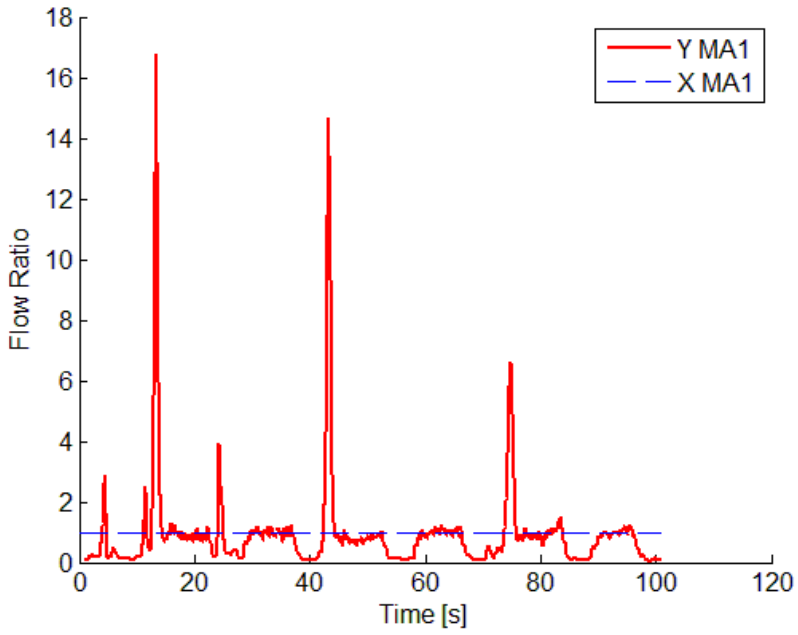


Fig. 7.7.1. Ratio of y- to x-flow integrated over a window of 30 frames (i.e. 1 second).

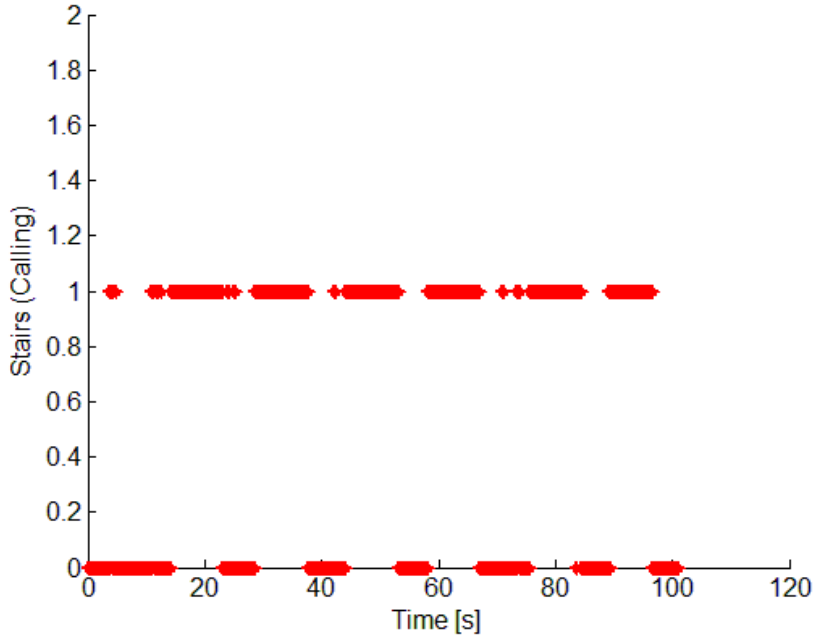


Fig. 7.7.2. Stairs indicator.

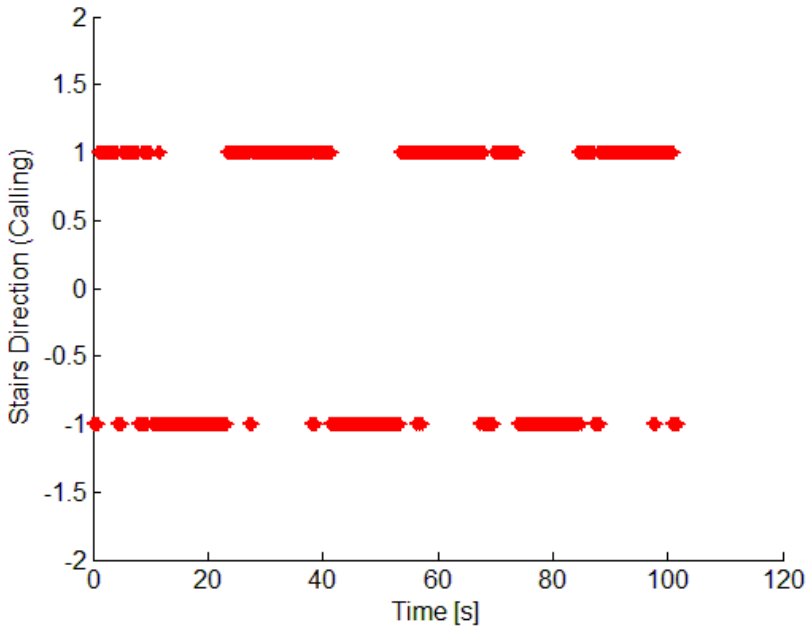


Fig. 7.7.3. Direction indicator.

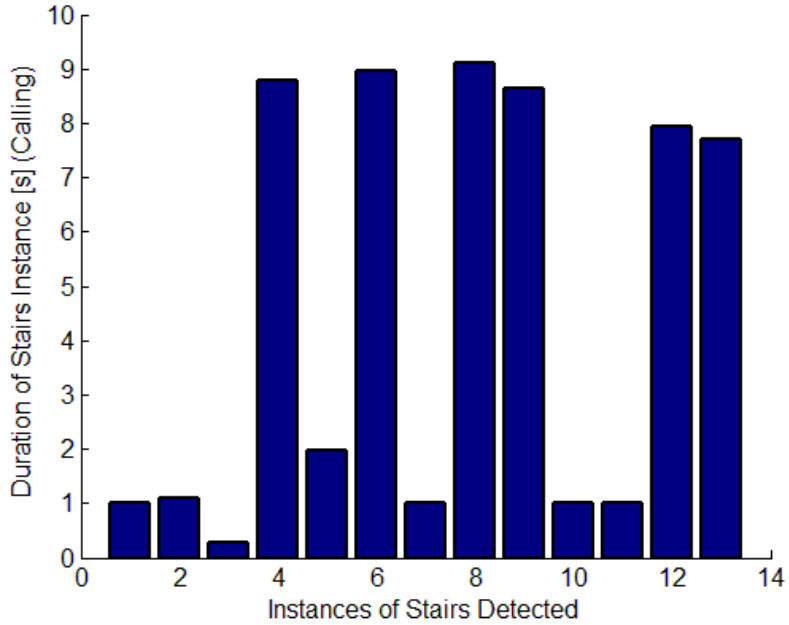


Fig. 7.7.4. Duration of detected instances of stairs.

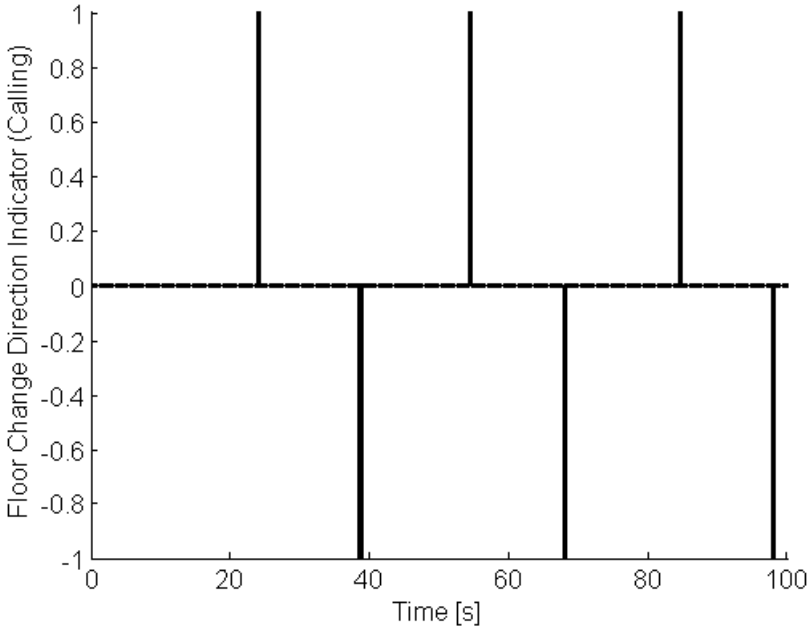


Fig. 7.7.5. Floor change indicator. A positive spike indicates a positive floor change while a negative spike indicates a negative floor change.

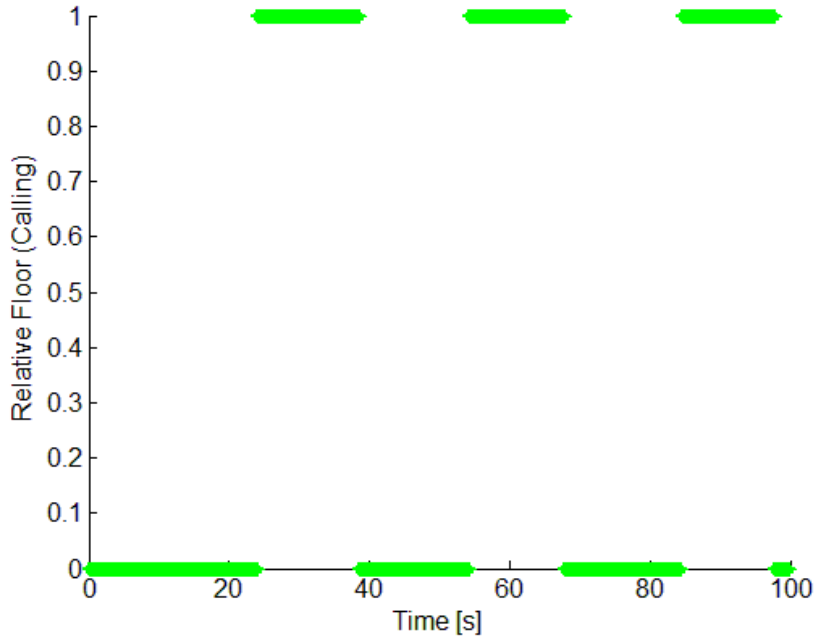


Fig. 7.7.6. Current floor.

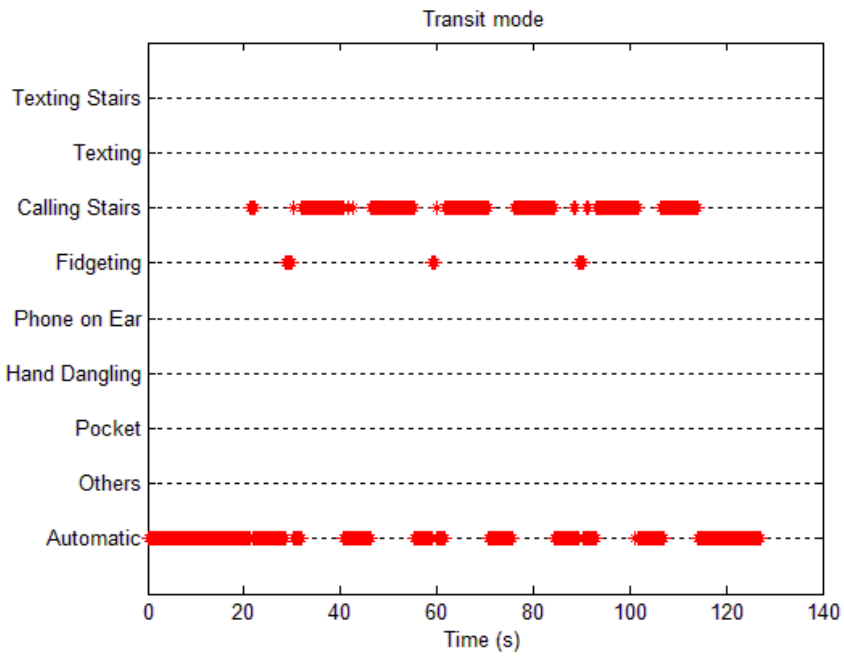
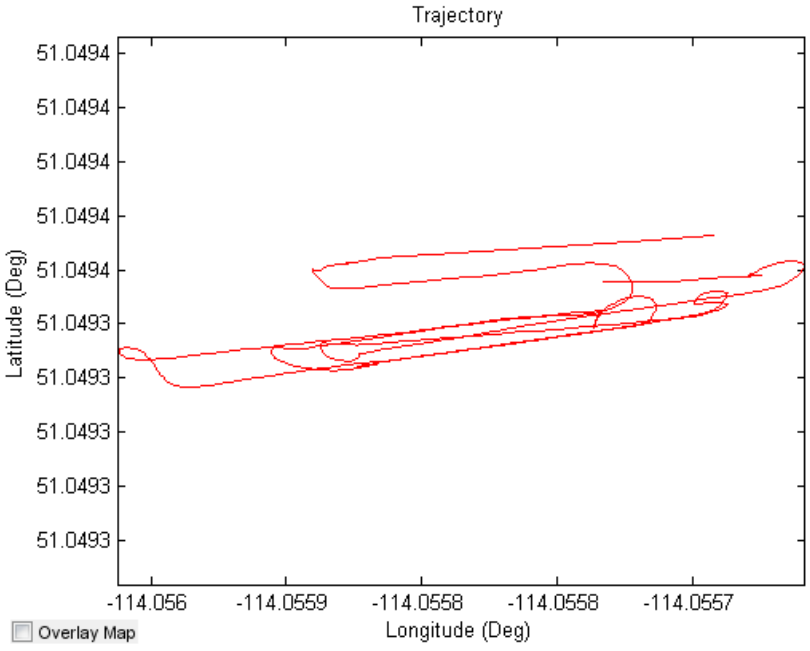
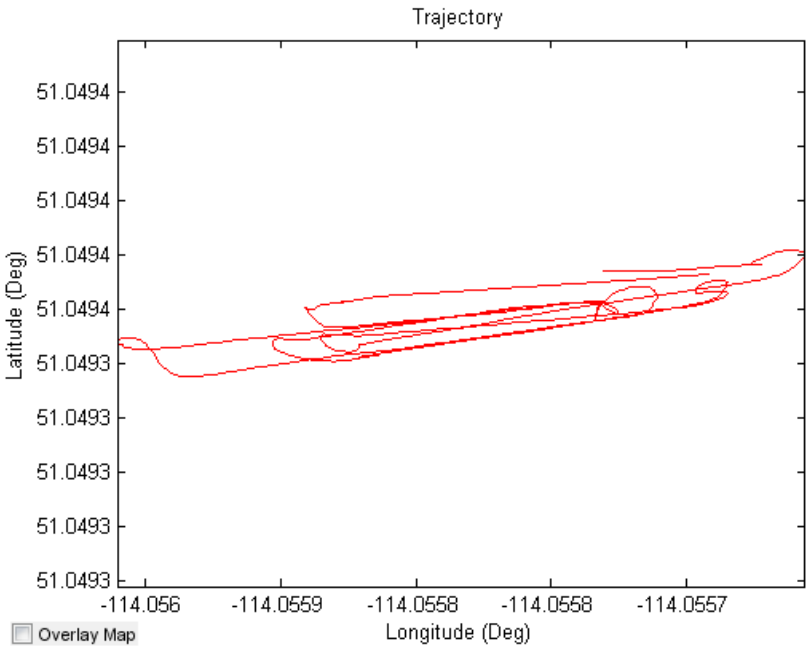


Fig. 7.7.7. Context classification obtained with vision aiding.

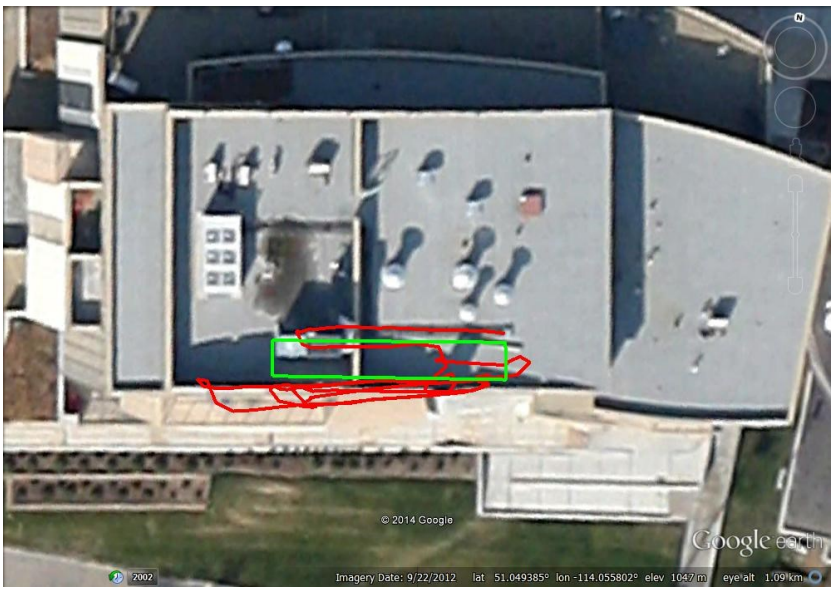


(a)

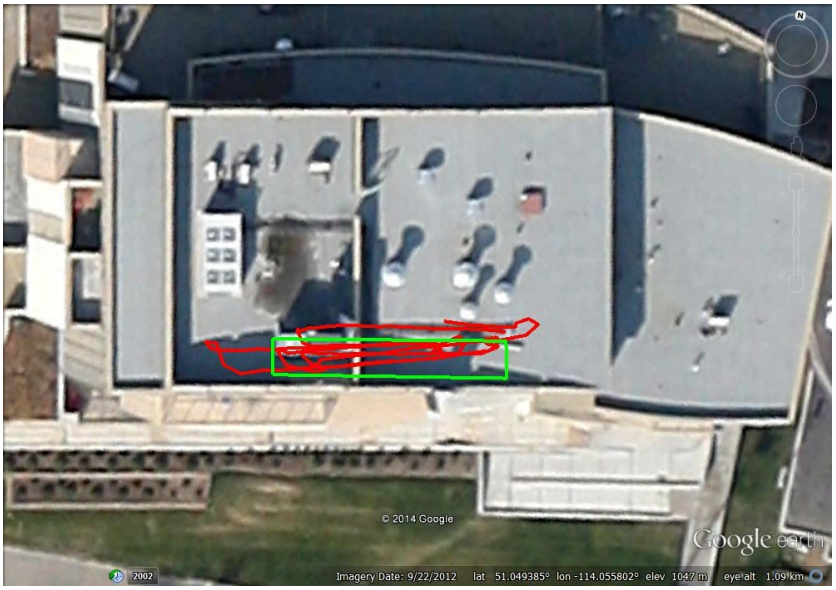


(b)

Fig. 7.7.8. Trajectory obtained (a) without context detection and (b) with vision-based context detection. The reference path is shown in green.



(a)



(b)

Fig. 7.7.9. Real-world trajectories obtained (a) without context detection (b) using vision-based context detection.

Chapter 8: Conclusion and Future Work

8.1 Conclusion

As stated in the research objectives in Chapter 1, the aim of this work was three-fold. The first aim was to develop a module to calculate the angle of a handheld portable navigation device with respect to the direction of its motion using a series of images captured by the device's built-in camera. To this end, images were captured and pre-processed before proceeding further. The pre-processing stage applied histogram equalization to mitigate changes in luminosity between frames and smoothing to remove high spatial frequency noise. The images were then converted to grayscale and used to calculate the optical flow between each pair. The optical flow calculation returned a flow map that mapped each pixel in the first image to its corresponding location in the second. Aggregating the flow map resulted in a tuple indicating the total amount of horizontal and vertical translation (in units of pixels) undergone by the first image to obtain the second. As the camera axis is aligned to the device axis, the arctangent of the ratio of vertical to horizontal translation in the image stream provided the device angle with great accuracy in the texting use case (i.e. when the device was facing the ground). The vertical use case (i.e. when the device was held upright) suffered from a divergent optical flow field and required the additional step of floor identification. The floor identification procedure was to partition the image into a 3x3 cell grid and use the cell in which the floor was expected to be found (based on the pitch and roll of the device) to perform the optical flow calculation and obtain the flow map.

The second aim was to develop a module to use parameters extracted from the aforementioned images to perform context classification. The aim of the classification was to distinguish whether the device was static or in motion ("static detection"), distinguish between meaningful and non-meaningful motion ("fidgeting detection"), and determine whether the device is undergoing

motion on a single floor in the texting or calling use case ("single floor texting/calling use case", respectively) or undergoing a relative change in height in either mode ("stairs texting/calling use case", respectively). To this end, the algorithm proceeded by performing a check on the MSSIM index between pairs of images. The MSSIM index is a value between 0 and 1 that reflects whether the images are completely different or identical, respectively. An MSSIM index value near unity would indicate that the images are nearly identical and that the device is stationary with the camera looking at the same scene. Any device motion of sufficient magnitude would alter the scene that the camera is looking at and lower the MSSIM. The MSSIM index failed to distinguish between a static feature-rich scene and a moving featureless scene, thereby necessitating a further check to determine whether the images were sufficiently feature-rich to proceed. The degree to which an image was feature-rich was dubbed the "Availability" of the image and quantified using the normalized value of the aggregated spatial derivative. With a sufficiently feature-rich image, the value of the MSSIM index could be used to distinguish whether the device was being held static or in motion.

With the device in motion, the value of the device angle variance over a small moving window was used to distinguish whether the device was engaged in meaningful motion or erratic fidgeting such as that undergone when texting or playing a game. When determined to be undergoing meaningful motion, the pitch, roll, and misalignment of the device were used to distinguish between the texting and calling use cases. In the texting use case, the Hough transform was used to examine the image stream for repeating lines perpendicular to the direction of motion that were indicative of the presence of stairs. In the presence and absence of such lines, the device was taken to be in the stairs and single floor texting use case, respectively. The images were also examined for lines parallel to the direction of motion. In the presence of repeating perpendicular and parallel

lines, the device is able to distinguish motion on a tile-based floor pattern. In the calling use case, the ratio of the levelled vertical and horizontal aggregated optical flow values after inter integration over a small window in time was used to differentiate between the stairs and single floor texting use cases. The ratio of vertical to horizontal flow was found to be slightly above unity in the stairs calling use case and significantly below unity in the single floor calling use case.

8.2 Thesis Contribution

Vision aiding has been shown to successfully provide estimation of the device heading misalignment as well as the device context. Accurate automatic misalignment estimation has a significant impact on the positioning solution of a PDR-based smartphone navigation system in terms of positional accuracy and can serve to reduce the set of potential failure cases. Furthermore, vision aiding has been shown to be capable of providing height updates in the form of floor changes. The main contribution of this thesis can be summarized as follows:

1. The development of a software module that that can successfully calculate the angle of a handheld portable navigation device with respect to the direction of its motion using a series of images captured by the device's built-in camera. This angle is known as the device heading misalignment and is used to determine the correct platform heading to be used in the navigation equations. Though similar work has been done using optical flow with feature-based methods [58], the direct method employed in this work using the image intensity values has the advantage of being less prone to outliers and more robust in scenes that are lacking in many prominent features [59]. An increased number of use cases are also addressed.

2. The development a software module that can use parameters extracted from the aforementioned images to perform context classification. The classified contexts include: whether the device is static or in motion ("static detection"); whether the device is engaging in navigationally meaningful or non-meaningful motion ("fidgeting detection"); and whether the device is undergoing motion on a single floor in a texting or calling use case ("single floor texting use case" or "single floor calling use case", respectively) or undergoing a relative change in height in either use case ("stairs texting use case" or "stairs calling use case", respectively). These contexts are used to improve the navigation solution. For example, the system can maintain the previously computed position when the device is determined to be static or fidgeting. In the case of PDR, the system can reduce the stride length in the presence of stairs and obtain otherwise missing information regarding height changes.
3. The encapsulation of the aforementioned modules into a single software entity that can be used to update a navigation system for hand-held devices in real-time.

8.3 Future Work

In this thesis, inertial data and video were captured simultaneously from the device in various scenarios and processed post-mission using a PDR emulator and offline implementation of the vision aiding module on a host PC. Though vision aiding showed improvements to the solution when processed offline, it is recommended to implement the vision aiding algorithm on a mobile device in order to assess the practicality of using a vision sensor in a scenario where battery life is a scarce resource. Implementation in a mobile environment will certainly impact the nature of the algorithms used in the various stages and lead to a more efficient design. For example, the image

gradient calculated to determine the feature-richness of the image in the Availability Determination routine can be re-used for edge detection in the context classification routine. Many optimizations can be made in this manner.

Additionally, it is recommended that optimal tuning of the parameters and choice of algorithms in the various stages can be explored. In this thesis, Gaussian smoothing is used in in the pre-processing stage but other methods such as median or bilateral smoothing can be assessed for performance enhancement. As another example, the optical flow calculation algorithm used in the current implementation is Farnebeck optical flow. It is recommended that the Camus [60], Horn and Schunck [61], Lucas and Kanade [62], and Nagel [63] optical flow algorithms be assessed for performance. Each has a variety of parameters that can be tuned for more accurate calculations.

Handheld devices are used in a myriad of contexts and detecting more than a few lay beyond the scope of this work. It is recommended that researchers mine the image stream to classify additional device contexts, for example whether the device is undergoing motion on an escalator or a moving sidewalk.

References

[1] El-Sheimy, N. (2006). "ENGO 623: Inertial Techniques and INS/DGPS Integration", Lecture notes, Department of Geomatics Engineering, University of Calgary, Canada.

[2] Noureldin, A., Karamat, T. B., & Georgy, J. (2013). "Chapter 4: Inertial Navigation System" in *Fundamentals of inertial navigation, satellite-based positioning and their integration*. Berlin/Heidelberg, Germany: Springer, pp. 125-166.

[3] Noureldin, A., Karamat, T. B., & Georgy, J. (2013). "Chapter 5: Inertial Navigation System Modelling" in *Fundamentals of inertial navigation, satellite-based positioning and their integration*. Berlin/Heidelberg, Germany: Springer, pp. 167-200.

[4] VectorNav, "Inertial Sensor Market", http://www.vectornav.com/support/library/inertial-sensor-market, Visited May 2014.

[5] Stirling, R., Fyfe, K., & Lachapelle, G. (2005). "Evaluation of a new method of heading estimation for pedestrian dead reckoning using shoe mounted sensors", *Journal of Navigation*, *58*(1), pp. 31-45.

[6] Levi, R. W., & Judd, T. (1996). "Dead reckoning navigation system using accelerometer to measure foot impacts", US Patent 5,583,776.

[7] Kappi, J., Syrjarinne, J., & Saarinen, J. (2001). "MEMS-IMU based pedestrian navigator for handheld devices", *Proceedings of the 14th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS 2001)*, pp. 1369-1373.

[8] Mautz, R., & Tilch, S. (2011). "Survey of optical indoor positioning systems", *2011 International Conference on Positioning and Indoor Navigation (IPIN),* pp. 1-7.

[9] Hile, H., & Borriello, G. (2008). "Positioning and orientation in indoor environments using camera phones", *IEEE Computer Graphics and Applications*, *28*(4), pp. 32-39.

[10] Huang, B. (2013). “Floor plan based indoor vision navigation using smart device”, MSc Thesis, University of Calgary, UCGE Report 20380.

[11] Ido, J., Shimizu, Y., Matsumoto, Y., & Ogasawara, T. (2009). “Indoor navigation for a humanoid robot using a view sequence”, *The International Journal of Robotics Research*, *28*(2), pp. 315-325.

[12] Mulloni, A., Wagner, D., Schmalstieg, D. and Barakonyi, I. (2009). “Indoor positioning and navigation with camera phones”, *IEEE Transactions on Pervasive Computing*, Vol. 8, pp. 22-31.

[13] Tilch, S., & Mautz, R. (2011). "CLIPS proceedings", *Indoor Positioning and Indoor Navigation (IPIN), 2011 IEEE International Conference on*.

[14] Bürki, B., Guillaume, S., Sorber, P. and Oesch, H. (2010). “DAEDALUS: A Versatile Usable Digital Clip-on Measuring System for Total Stations”, *Proceedings of IPIN 2010*, Zurich, Switzerland, pp. 32-41.

[15] Aufderheide, D. and Krybus, W. (2010). “Towards real-time camera egomotion estimation and three-dimensional scene acquisition from monocular image streams”, *Proceedings of IPIN 2010*, Zurich, Switzerland, pp. 13-22.

[16] Ruotsalainen, L., Bancroft, J., & Lachapelle, G. (2012). “Mitigation of attitude and gyro errors through vision aiding”, In *International Conference on Indoor Positioning and Indoor Navigation*, Vol. 13.

[17] Ruotsalainen, L., Bancroft, J., Lachapelle, G., Kuusniemi, H., & Chen R. (2012). “Effect of

camera characteristics on the accuracy of a visual gyroscope for indoor pedestrian navigation", *Second International Conference on Ubiquitous Positioning, Indoor Navigation and Location-Based Service*s, Helsinki, Finland.

[18] Güzel, M. S. (2013). "Autonomous vehicle navigation using vision and mapless strategies: a survey", *Advances in Mechanical Engineering*.

[19] Bonin-Font, F., Ortiz, A., & Oliver, G. (2008). "Visual navigation for mobile robots: A survey", *Journal of intelligent and robotic systems*, *53*(3), pp. 263-296.

[20] Dellaert, F., Fox, D., Burgard, W., & Thrun, S. (1999). "Monte carlo localization for mobile robots", *Robotics and Automation, Proceedings of the 1999 IEEE International Conference on,* Vol. 2, pp. 1322-1328.

[21] Davison, A. J. (2003). "Real-time simultaneous localisation and mapping with a single camera", *Computer Vision, Proceedings of the Ninth IEEE International Conference on*, pp. 1403-1410.

[22] Angermann, M., & Robertson, P. (2012). "Footslam: Pedestrian simultaneous localization and mapping without exteroceptive sensors—hitchhiking on human perception and cognition", *Proceedings of the IEEE*, pp. 1840-1848.

[23] Silveira, G., Malis, E., & Rives, P. (2008). "An efficient direct approach to visual SLAM", *Robotics, IEEE Transactions on*, *24*(5), pp. 969-979.

[24] Smith, R. C., & Cheeseman, P. (1986). "On the representation and estimation of spatial uncertainty", *The International Journal of Robotics Research*, *5*(4), pp. 56-68.

[25] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. (2003). "FastSLAM 2.0: An

improved particle filtering algorithm for simultaneous localization and mapping that provably converges," *Artificial Intelligence, Proceedings of the International Joint Conference on*, pp. 1151–1156.

[26] Lowe, D.G. (1999). "Object recognition from local scale invariant features", *Proceedings of the International Conference on Computer Vision (ICCV)*, pp. 1150–1157.

[27] Bay, H., Tuytelaars, T., & Van Gool, L. (2006). "SURF: Speeded up robust features", *Computer Vision–ECCV 2006*, pp. 404-417.

[28] Knight, J., Davison, A., & Reid, I. (2001). "Towards constant time SLAM using Postponement", *Intelligent Robots and Systems, Proceedings of the 2001 IEEE/RSJ International Conference on*, Vol. 1, pp. 405-413.

[29] Davison, A. J., Cid, Y. G., & Kita, N. (2004). "Real-time 3D SLAM with wide-angle vision", *Proceedings of the IFAC/EURON Symposium on Intelligent Autonomous Vehicles*.

[30] Hérissé, B., Hamel, T., Mahony, R., & Russotto, F. X. (2012). "Landing a VTOL unmanned aerial vehicle on a moving platform using optical flow", *Robotics, IEEE Transactions on*, *28*(1), pp. 77-89.

[31] Bernardino, A., & Santos-Victor, J. (1997). "Visual behaviours for binocular tracking", *Advanced Mobile Robots, Proceedings of the Second EUROMICRO Workshop on*, pp. 2-7.

[32] Talukder, A., Goldberg, S., Matthies, L., & Ansar, A. (2003). "Real-time detection of moving objects in a dynamic scene from moving robotic vehicles", *Intelligent Robots and Systems (IROS 2003), Proceedings of the 2003 IEEE/RSJ International Conference on*, Vol. 2, pp. 1308-1313.

[33] DeSouza, G. N., & Kak, A. C. (2002). "Vision for mobile robot navigation: A

survey", *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *24*(2), pp. 237-267.

[34] Matsumoto, Y., Ikeda, K., Inaba, M., & Inoue, H. (1999). "Visual navigation using omnidirectional view sequence", *Intelligent Robots and Systems (IROS '99). Proceedings of the 1999 IEEE/RSJ International Conference on*, Vol. 1, pp. 317-322.

[35] Matsumoto, Y., Sakai, K., Inaba, M., & Inoue, H. (2000). "View-based approach to robot navigation", *Intelligent Robots and Systems (IROS 2000), Proceedings of the 2000 IEEE/RSJ International Conference on*, Vol. 3, pp. 1702-1708.

[36] Courbon, J., Mezouar, Y., Guénard, N., & Martinet, P. (2010). "Vision-based navigation of unmanned aerial vehicles", *Control Engineering Practice*, *18*(7), pp. 789-799.

[37] Gaspar, J., Winters, N., & Santos-Victor, J. (2000). "Vision-based navigation and environmental representations with an omnidirectional camera", *IEEE Transactions on Robotics and Automation 16*(6), pp. 890-898.

[38] Hrabar, S., & Sukhatme, G. (2004). "A comparison of two camera configurations for optic-flow based navigation of a UAV through urban canyons", *Intelligent Robots and Systems (IROS 2004), Proceedings of the 2004 IEEE/RSJ International Conference on*, Vol. 3, pp. 2673-2680.

[39] Lorigo, L. M., Brooks, R. A., & Grimsou, W. E. L. (1997). "Visually-guided obstacle avoidance in unstructured environments", *Intelligent Robots and Systems (IROS '97), Proceedings of the 1997 IEEE/RSJ International Conference on,* Vol. 1, pp. 373-379.

[40] Howard, A., Tunstel, E., Edwards, D., & Carlson, A. (2001). "Enhancing fuzzy robot

navigation systems by mimicking human visual perception of natural terrain traversability", *Joint 9th IFSA World Congress and 20th NAFIPS International Conference,* Vol. 1, pp. 7-12.

[41] Kazemipur, B., Syed, Z., Georgy, J., El-Sheimy, N. (2013). "Vision-based real-time smartphone heading and misalignment", *Proceedings of the 26th International Technical Meeting of The Satellite Division of the Institute of Navigation* (*ION GNSS+ 2013*) 505-510.

[42] Johansson, B. and Farnebäck, G. (2002). "A theoretical comparison of different orientation tensors", *Proceedings of the SSAB02 Symposium on Image Analysis*, pp. 69-73.

[43] ITU-RRecommendation BT.601, "Encoding parameters of digital television for studios", Recommendations of the ITU Radiocommunication Sector.

[44] Trahanias, P. E., & Venetsanopoulos, A. N. (1992). "Color image enhancement through 3-D histogram equalization", *Pattern Recognition (IAPR '92), Proceedings of the 11th International Conference on*, Vol. 3, pp. 545-548.

[45] Ying-Dong, Q., Cheng-Song, C., San-Ben, C., & Jin-Quan, L. (2005). "A fast subpixel edge detection method using Sobel–Zernike moments operator", *Image and Vision Computing*, *23*(1), pp. 11-17.

[46] Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). "Image quality assessment: from error visibility to structural similarity", *Image Processing, IEEE Transactions on*, *13*(4), pp. 600-612.

[47] Farnebäck, G. (2000). "Orientation estimation based on weighted projection onto quadratic polynomials", *VMV*, pp. 89-96.

[48] Farnebäck, G. (2000). "Fast and accurate motion estimation using orientation tensors and

parametric motion models", *Pattern Recognition. Proceedings of the 15th International Conference on*, Vol. 1, pp. 135-139.

[49] Kazemipur, B., Syed, Z., Georgy, J., El-Sheimy, N. (2014). "Vision-based context and height estimation for 3D indoor location", *Proceedings of the IEEE/ION Position, Location, and Navigation Symposium 2014 (IEEE/ION PLANS 2014), Monterey, CA*, pp. 1336-1342.

[50] Otieno, B. S. (2002). "An alternative estimate of preferred direction for circular data", Doctoral dissertation, Virginia Polytechnic Institute and State University**.**

[51] Se, S., & Brady, M. (2000). "Vision-based detection of staircases", *Fourth Asian Conference on Computer Vision ACCV,* Vol. 1, pp. 535-540.

[52] Hesch, J. A., Mariottini, G. L., & Roumeliotis, S. I. (2010). "Descending-stair detection, approach, and traversal with an autonomous tracked vehicle", *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on,* pp. 5525-5531.

[53] Canny, J. (1986). "A computational approach to edge detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence 8*(6), pp. 679–698.

[54] Ballard, D. (1981). "Generalizing the Hough transform to detect arbitrary shapes", *Pattern Recognition 13*(2), pp. 111-122.

[55] Solberg, A. (2009). "INF 4300 – Digital Image Analysis." Lecture Notes. Faculty of Mathematics and Natural Sciences, University of Oslo. Lecture notes obtained from URL: http://www.uio.no/studier/emner/matnat/ifi/INF4300/h09/undervisningsmateriale/hough09.pdf

[56] Kazemipur, B., Syed, Z., Georgy, J., El-Sheimy, N. (2014). "Real-time Vision-aiding for

Reliable 3D Indoor Location," *Proceedings of the 27th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2014)*, Tampa, FL, pp. 2118-2131.

[57] Kazemipur, B., Georgy, J., Syed, Z., El-Sheimy, N. "Misalignment and context for portable navigation using computer vison," *IEEE Transactions on Image Processing*. Submitted for publication December 2014.

[58] Saeedi, S., Moussa, A., & El-Sheimy, N. (2014). "Context-aware personal navigation using embedded sensor fusion in smartphones", *Sensors*, *14*(4), pp. 5742-5767.

[59] Schops, T., Enge, J., & Cremers, D. (2014). "Semi-dense visual odometry for AR on a smartphone", *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 145-150.

[60] Camus, T. (1995). "Real-time quantized optical flow." *Computer Architectures for Machine Perception (CAMP '95), Proceedings of*.

[61] Horn, B. K., & Schunck, B. G. (1981). "Determining optical flow", *International Society for Optics and Photonics 1981 Technical Symposium*, pp. 319-331.

[62] Lucas, B. D., & Kanade, T. (1981). "An iterative image registration technique with an application to stereo vision", *International Joint Conferences on Artificial Intelligence*, Vol. 81, pp. 674-679.

[63] Nagel, H. H. (1987). "On the estimation of optical flow: Relations between different approaches and some new results", *Artificial Intelligence*, *33*(3), pp. 299-324.