MORGAN & CLAYPOOL PUBLISHERS

Sensor Analysis for the Internet of Things

Michael Stanley
Jongmin Lee

Synthesis Lectures on Algorithms and Software in Engineering

Andreas Spanias, *Series Editor*

Sensor Analysis for the Internet of Things

Synthesis Lectures on Algorithms and Software in Engineering

Editor
Andreas Spanias, ***Arizona State University***

Sensor Analysis for the Internet of Things
Michael Stanley and Jongmin Lee
2017

Virtual Design of an Audio Lifelogging System: Tools for IoT Systems
Brian Mears and Mohit Shah
2016

Despeckle Filtering for Ultrasound Imaging and Video, Volume II: Selected Applications, Second Edition
Christos P. Loizou and Constantinos S. Pattichis
2015

Despeckle Filtering for Ultrasound Imaging and Video, Volume I: Algorithms and Software, Second Edition
Christos P. Loizou and Constantinos S. Pattichis
2015

Latency and Distortion of Electromagnetic Trackers for Augmented Reality Systems
Henry Himberg and Yuichi Motai
2014

Bandwidth Extension of Speech Using Perceptual Criteria
Visar Berisha, Steven Sandoval, and Julie Liss
2013

Control Grid Motion Estimation for Efficient Application of Optical Flow
Christine M. Zwart and David H. Frakes
2013

Sparse Representations for Radar with MATLAB ™ Examples
Peter Knee
2012

Analysis of the MPEG-1 Layer III (MP3) Algorithm Using MATLAB
Jayaraman J. Thiagarajan and Andreas Spanias
2011

Theory and Applications of Gaussian Quadrature Methods
Narayan Kovvali
2011

Algorithms and Software for Predictive and Perceptual Modeling of Speech
Venkatraman Atti
2011

Adaptive High-Resolution Sensor Waveform Design for Tracking
Ioannis Kyriakides, Darryl Morrell, and Antonia Papandreou-Suppappola
2010

MATLAB™ Software for the Code Excited Linear Prediction Algorithm: The Federal Standard-1016
Karthikeyan N. Ramamurthy and Andreas S. Spanias
2010

OFDM Systems for Wireless Communications
Adarsh B. Narasimhamurthy, Mahesh K. Banavar, and Cihan Tepedelenliouglu
2010

Advances in Modern Blind Signal Separation Algorithms: Theory and Applications
Kostas Kokkinakis and Philipos C. Loizou
2010

Advances in Waveform-Agile Sensing for Tracking
Sandeep Prasad Sira, Antonia Papandreou-Suppappola, and Darryl Morrell
2008

Despeckle Filtering Algorithms and Software for Ultrasound Imaging
Christos P. Loizou and Constantinos S. Pattichis
2008

Sensor Analysis for the Internet of Things

Michael Stanley and Jongmin Lee

www.morganclaypool.com

ISBN: 9781681732879 paperback
ISBN: 9781681732886 ebook
ISBN: 9781681732893 hardcover

DOI 10.2200/S00827ED1V01Y201802ASE017

A Publication in the Morgan & Claypool Publishers series
SYNTHESIS LECTURES ON ALGORITHMS AND SOFTWARE IN ENGINEERING

Lecture #17
Series Editor: Andreas Spanias, *Arizona State University*
Series ISSN
Print 1938-1727 Electronic 1938-1735

Sensor Analysis for the Internet of Things

Michael Stanley and Jongmin Lee
NXP Semiconductor

SYNTHESIS LECTURES ON ALGORITHMS AND SOFTWARE IN ENGINEERING #17

MORGAN & CLAYPOOL PUBLISHERS

ABSTRACT

While it may be attractive to view sensors as simple transducers which convert physical quantities into electrical signals, the truth of the matter is more complex. The engineer should have a proper understanding of the physics involved in the conversion process, including interactions with other measurable quantities. A deep understanding of these interactions can be leveraged to apply sensor fusion techniques to minimize noise and/or extract additional information from sensor signals.

Advances in microcontroller and MEMS manufacturing, along with improved internet connectivity, have enabled cost-effective wearable and Internet of Things sensor applications. At the same time, machine learning techniques have gone mainstream, so that those same applications can now be more intelligent than ever before. This book explores these topics in the context of a small set of sensor types.

We provide some basic understanding of sensor operation for accelerometers, magnetometers, gyroscopes, and pressure sensors. We show how information from these can be fused to provide estimates of orientation. Then we explore the topics of machine learning and sensor data analytics.

KEYWORDS

sensors, accelerometer, gyroscope, magnetometer, pressure, Internet of Things (IoT), algorithms, sensor fusion, machine learning, deep learning

Contents

List of Figures

List of Tables

Preface

This book is a product of collaboration between the Sensor, Signal & Information Processing (SenSIP) center at Arizona State University and NXP Semiconductor's sensor division. SenSIP is one of the sites of the Industry/University Cooperative Research Center (I/UCRC) for Net-Centric Software and Systems, partially sponsored by the National Science Foundation and industrial members. Freescale Semiconductor (now a part of NXP) joined SenSIP in 2014, and the two organizations began a joint project to explore the use of sensors and machine learning techniques in the context of machine condition monitoring.

Student investigators Jongmin Lee (now Dr. Lee, and an NXP engineer) and Uday Shankar Shanthamallu worked under the supervision of SenSIP director Dr. Andreas Spanias and Michael Stanley at NXP to evaluate the application of standard machine learning techniques to the embedded design space. We were specifically interested in bringing machine learning techniques to the IoT "leaf node".

Prior to joining SenSIP, Stanley, Mark Pedley, and Dr. Zbigniew Baranski of NXP spent significant efforts developing and documenting what is now the open-sourced NXP Sensor Fusion Library. Stanley also wrote a series of blogs on sensor and sensor fusion topics.

As we gained additional insights during the SenSIP project, we began to consider pulling materials from that project, as well as our existing blogs and sensor fusion documentation for an overview book on topics of sensors for IoT and machine learning. This book is a result of that effort. It is intended to provide a sampling of sensor and sensor data analysis topics, with sufficient references to allow the interested user to follow up on specific areas of interest.

Michael Stanley and Jongmin Lee
December 2017

Acknowledgments

The authors have been supported in this work by NXP Semiconductors N.V. Many of the figures in this text were reprinted with permission of NXP Semiconductors.

Michael Stanley and Jongmin Lee
December 2017

Nomenclature

Symbols

$\mathcal{A}$	=	reference frame
$\mathcal{B}$	=	reference frame
Pa	=	pascal, the SI unit of pressure
X	=	X axis
Y	=	Y axis
Z	=	Z axis
P	=	position
$\mathbb{R}^3$	=	3D Euclidean space
q	=	quaternion
$N(q)$	=	norm of quaternion q
$\times$	=	the cross product
$\cdot$	=	the dot product
$\perp$	=	orthogonal
$\mathcal{G}$	=	global reference frame
$\mathcal{S}$	=	sensor reference frame
$\Phi(\boldsymbol{x})$	=	nonlinear mapping function of $\boldsymbol{x}$ in higher-dimensional space
$\kappa(\cdot,\cdot)$	=	kernel function
C_1	=	class 1
C_{-1}	=	class -1
$\boldsymbol{M}^{\mathrm{T}}$	=	matrix transpose

Scalars

T	=	Tesla
G	=	Gauss
$\|\boldsymbol{B}\|$	=	magnitude of magnetic field
m	=	mass
p	=	a pressure
q_0	=	scalar part of quaternion q
μ_0	=	magnetic permeability of free space

Vectors and Matrices

$\boldsymbol{a}_1$	=	first basis vector in reference frame $\mathcal{A}$
$\boldsymbol{a}_2$	=	second basis vector in reference frame $\mathcal{A}$
$\boldsymbol{a}_3$	=	third basis vector in reference frame $\mathcal{A}$
$\boldsymbol{b}_1$	=	first basis vector in reference frame $\mathcal{B}$
$\boldsymbol{b}_2$	=	second basis vector in reference frame $\mathcal{B}$
$\boldsymbol{b}_3$	=	third basis vector in reference frame $\mathcal{B}$
$\boldsymbol{a}$	=	a general vector
$\boldsymbol{b}$	=	a general vector
$\boldsymbol{r}$	=	a position offset vector
$\boldsymbol{x}$	=	a general vector
$\boldsymbol{\Sigma}$	=	a covariance matrix
$\boldsymbol{\mu}$	=	a mean vector
${}^{\mathcal{A}}\frac{d\boldsymbol{r}}{dt}$	=	rate of change of a vector $\boldsymbol{r}$ as viewed in reference frame $\mathcal{A}$
$\frac{dr_1}{dt}$	=	rate of change of a scalar r_1
${}^{\mathcal{A}}\boldsymbol{\omega}^{\mathcal{B}}$	=	angular velocity of reference frame $\mathcal{B}$ as viewed in reference frame $\mathcal{A}$
${}^{\mathcal{B}}\frac{d^2\boldsymbol{r}}{dt^2}$	=	acceleration of a vector $\boldsymbol{r}$ as viewed in reference frame $\mathcal{B}$
$\boldsymbol{f}$	=	a force vector

$\boldsymbol{i}$	=	standard orthonormal basis in $\mathbb{R}^3$
$\boldsymbol{j}$	=	standard orthonormal basis in $\mathbb{R}^3$
$\boldsymbol{k}$	=	standard orthonormal basis in $\mathbb{R}^3$
$\boldsymbol{q}$	=	vector part of quaternion in $\mathbb{R}^3$
$\boldsymbol{M}_\psi$	=	rotation matrix for rotation angle ψ
$\boldsymbol{M}$	=	a general rotation matrix
$\boldsymbol{M}^T$	=	transpose of a matrix $\boldsymbol{M}$
$\boldsymbol{M}^{-1}$	=	inverse of a matrix $\boldsymbol{M}$
$[\cdot]_\times$	=	skew-symmetric matrix
$\boldsymbol{I}$	=	identity matrix
$\boldsymbol{Q}$	=	origin point in a reference frame

CHAPTER 1

Introduction

As an increasing number of physical objects are connected to the Internet, the traditional concept of the Web has been extended to the *Internet of Things* (IoT) [122, 123, 124, 125]. Humans *and* devices such as sensors, actuators, radio-frequency identification (RFID), and smartphones can now communicate with each other, and to cloud-based resources. The emergence of the IoT has improved the quality of life in domains such as medicine, healthcare, automotive, industrial, energy, and so on. Figure 1.1 illustrates use cases of sensors in IoT applications, but there are many more applications.

The IoT enables the *things* in our lives to observe, understand, and control by sharing information and coordinating smart decisions. Development of the IoT has pushed the state-of-the-art across multiple technologies including sensors, communications and networks, data processing, cloud-based applications, user interfaces, security, and privacy. This book focuses

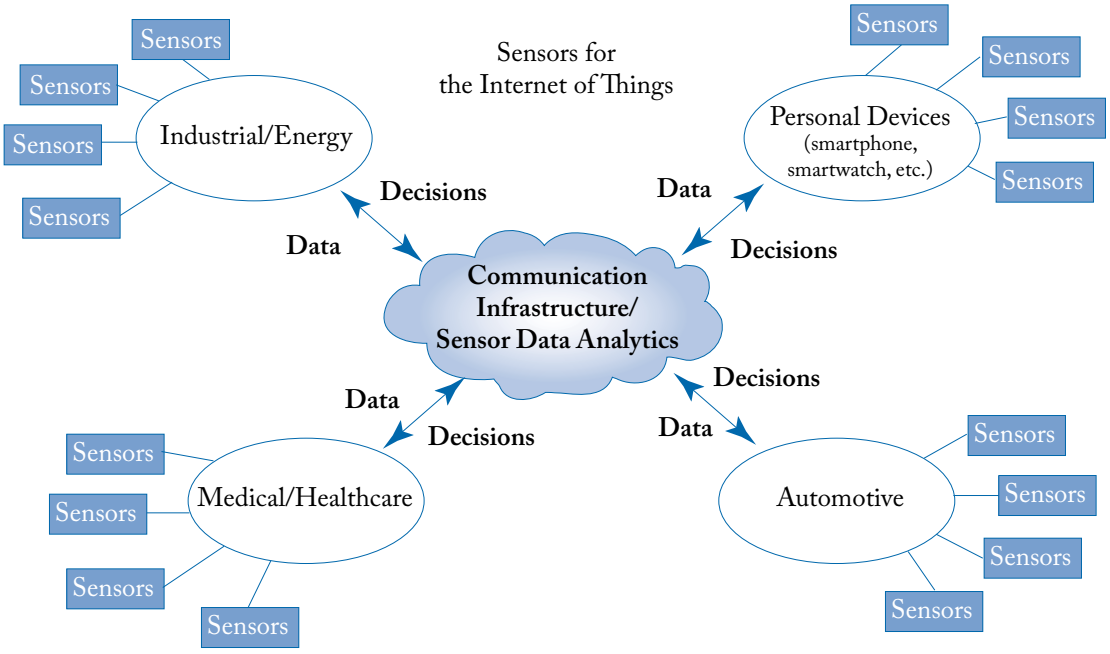

Figure 1.1: Sensors for the Internet of Things.

on sensor-related technical issues, with additional emphasis on sensor-generated data and its analytics. As sensor technologies continue to mature, the cost of various sensor types is being lowered so that they can be pervasively and cost-effectively deployed almost everywhere. A single smartphone incorporates many sensors including touch, inertial, sound, vision, and GPS. Modern vehicles may include multiple cameras, Lidar, pressure, and inertial sensors. Medical devices provide diagnostic information generated by various sensors. Transportation, road, and city managers can control traffic, and optimize city resources based on information observed by sensor devices.

Sensors transform physical quantities into signals which can be processed electronically. For instance, a digital accelerometer transforms gravity and linear acceleration into numerical values which can be consumed by a microcontroller. It is one type of *inertial* sensor. Most recent smartphones or game controllers include inertial sensors to detect and track acceleration, vibration, orientation, and other types of motion. Since accelerometers can measure vibrations of a physical object, they have been used for anomaly detection of industrial machinery where predictive maintenance (e.g., fault detection and part repair) is critical for cost-savings and safety. A magnetometer detects the ambient magnetic field. In the absence of magnetic interference, the earth's magnetic field varies depending upon position around the earth, but is almost constant for a given location. Rotating a 3D-magnetometer changes the frame of reference used for measurement, and results in corresponding changes to X/Y/Z electrical signals. This can be used to infer orientation of the sensor. Pressure and temperature sensors, respectively, transform the difference of environmental phenomena such as air pressure and temperature into signals. Pressure sensors are widely used to warn of under-inflated tires on vehicles, to estimate altitude as a function of air pressure, and in industrial process control loops.

A notable example of the explosion of sensors is the smartphone. Table 1.1 shows that more and more sensors have been integrated into smartphones over time. Exponentially larger sensor data streams are being collected and processed to provide us with more intelligent decisions and make our lives more convenient. Accelerometers and magnetometers have been included since the introduction of the original smartphones. These two sensors have been used in many other applications, including machine condition monitoring [1, 2], human body activities [3, 4, 5, 6, 7, 8], healthcare [9, 10, 11], and structural monitoring [12, 13]. In Chapter 2, we will study these two sensors in more detail. We will also explore two other sensors found in later generation phones: pressure sensors and gyroscopes.

While accurate measurement of physical phenomena by individual sensors is important, it is often insufficient to fully describe a problem. Individual sensor types have their own limitations. For instance, accelerometers are affected by both linear acceleration and gravity. Without additional sensors or assumptions, it is impossible to separate the two physical attributes. By combining data from multiple sensors, we can deduce information that cannot be gleaned from any individual device. One such example is orientation estimation, which has been widely used in applications such as aerospace, robotics, gaming, sports, and navigation. A gyroscope

Table 1.1: Sensor growth in smartphones

Smart Phone Model[1]	Introduction Date	Sensors
iPhone	2007	Accelerometer, proximity sensor, ambient light sensor
iPhone 3GS	2009	Accelerometer, magnetometer, proximity sensor, ambient light sensor
iPhone 4S	2011	Accelerometer, magnetometer, gyroscope, proximity sensor, ambient light sensor
iPhone 6	2014	Accelerometer, magnetometer, gyroscope, barometer, proximity sensor, ambient light sensor, finger print scanner, NFC, etc.
Galaxy S	2010	Accelerometer, magnetometer, ambient light sensor
Galaxy S2	2011	Accelerometer, magnetometer, gyroscope, proximity sensor, ambient light sensor
Galaxy S3	2012	Accelerometer, magnetometer, gyroscope, pressure sensor, proximity sensor, ambient light sensor, RGB light sensor
Galaxy S5	2014	Accelerometer, magnetometer, gyroscope, pressure sensor, proximity sensor, ambient light sensor, RGB light sensor, humidity sensor, hall effect sensor, fingerprint sensor, temperature, etc.

[1] Resources were collected from Apple.com, Samsung.com, and Wikipedia.

measures angular velocity that can be integrated to estimate changes in orientation. But gyroscope data alone is not good enough to estimate orientation—we need another sensor(s) to provide the baseline for that integration. Accelerometer and/or magnetometer sensors can be used as they measure the gravity and the magnetic fields (respectively) of the earth, which are good references for orientation. Both accelerometer and magnetometer are subject to interference from other sources. If we include data from all three sensor types into the orientation computation, we end up with a much more robust estimate. Orientation estimation by sensor fusion based on quaternions has been used because of its stability and efficient computation [4, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]. This is done via a process of *sensor* fusion that will be described in Chapter 3.

As communication and network technologies have improved, it became easy for large amounts of sensor data to be transmitted and shared for further process or management. People can wirelessly access the Internet on their smartphones or mobile devices. Even from remote distances, wireless sensor networks allow people to obtain data for monitoring and control. According to a report published from International Data Corporation (IDC) in 2012 [24], all digital data created, replicated, and consumed in a year will reach 40 trillion gigabytes by the end of 2020, and machine-generated data will increase up to 42% by then. Much of machine-generated data is sensor data. The growing volume of data created from sensors provides significant information and associated opportunities for IoT applications and businesses. The huge volume of sensor data is collectively known as "big data," and analytics for that data is an integral component of the reality of IoT environments.

Sensor data analytics is the process of examining large sets of collected sensor data to uncover unknown correlations or patterns, monitor real-time conditions, and make predictive decisions. The ultimate goal is to provide users with predictive models for better decisions. Since sensors are usually on, collecting data in real time, the analytics may require sophisticated procedures for data acquisition, processing, and building statistical models [25]. Sensor data is often collected in real time from highly dynamic systems. Real-time sensor data can be expensive to store because of the huge volumes of data being generated. When data streams are transmitted and shared, data quality can be deteriorated due to compression, missed data, measurement noise, and so on. Real-time data analysis is necessary to deal with sensor data and to learn statistical features from them to build predictive models. Most analytics for sensor data is performed in the cloud, however intermediate data analysis can reduce the amount of data transmitted and improve the efficiency of sensor data analytics for the IoT. Distributed sensor hubs collect local sensor data, filter out redundant information, and transmit them to cloud for analysis. Decreasing prices for micro-electromechanical systems (MEMS) technologies, along with increased microcontroller compute power and large memory sizes, make it possible for embedded devices to perform simple analytics locally in a fully distributed way without transmitting raw sensor data [26]. We introduce sensor data analytics in Chapter 4 with a brief review of machine learning techniques [27, 28].

Figure 1.2 summarizes the major chapters of this book. In Chapter 2, several types of sensors will be introduced with basic operating principles. We primarily focus on accelerometers, magnetometers, gyro sensors, and pressure sensors, which are among the most commonly used in our everyday lives with strong connection to IoT applications. But there are many excellent sensor types that can be applied to various applications. A few sensor fusion examples with different combinations of accelerometers, magnetometers, and gyro sensors will be described in Chapter 3. We will discuss analytics for the data generated from sensors and sensor fusion, with a brief review of machine learning algorithms in Chapter 4. In Chapter 5, we will discuss examples of the IoT applications in various fields.

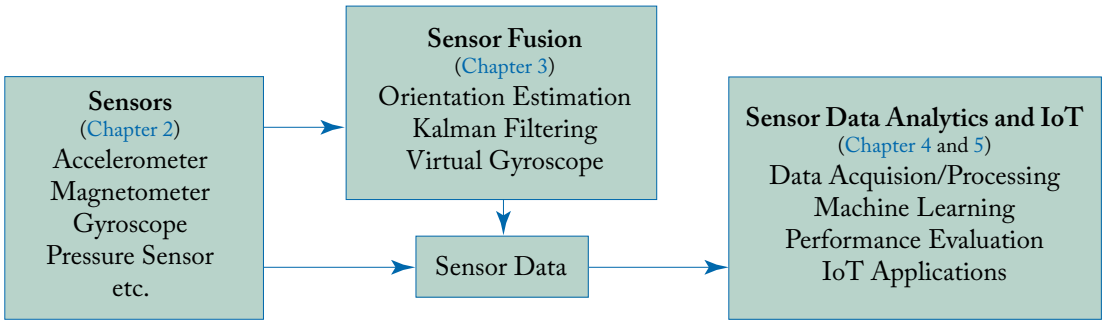

Figure 1.2: Summary of major chapters in this book.

CHAPTER 2

Sensors

Sensors convert physical properties into numerical quantities. There are typically many ways to measure any given physical property. Distance can be measured by using a yardstick, ultrasonic sensor, Lidar, and radar. Rotation can be estimated with accelerometers, magnetic sensors, and angular rate (gyro) sensors. These are a small subset of techniques used to measure just one physical quantity. To keep things simple, this book focuses on digital MEMS and magnetic sensors used to detect position, movement, and vibration. Specifically:

- accelerometers,
- magnetometers,
- gyroscopes, and
- pressure sensor/altimeters.

We are focusing on these sensor types because of their common use in orientation estimation systems, which is a common subsystem found in many products. But the lessons learned can be applied to a wide variety of sensor types.

When using any sensor, it is important to understand its limitations. These can come in many forms:

- limited resolution,
- noise (both environmental and in the sensor itself),
- effects of temperature,
- linearity,
- operating shifts with time, and
- sensitivity to other physical quantities.

Table 2.1 explores the effects of these limitations on our four key sensor types.

Table 2.1: Strengths and weaknesses of various sensor types

Sensor	Strengths	Weaknesses
Accelerometer	• Inexpensive • Extremely low power • Very linear • Very low noise	• Measures the sum of gravity and acceleration. We need them separate
Magnetometer	• The only sensor that can orient itself with regard to "North" • Insensitive to linear acceleration	• Subject to magnetic interference • Not "spatially constant"
Gyro Sensor	• Relatively independent of linear acceleration • Can be use to "gyro-compensate" the magnetometer	• Power hog • Relatively long startup time • Zero rate offset drifts over time
Pressure Sensor	• The only stand-alone sensor that can give an indication of altitude	• Not well understood • A "relative" measurement • Subject to many interferences and environmental factors • Require a highly precise Analog-to-Digital Converter (ADC) to obtain precise altitude estimates

2.1 ACCELEROMETER

The accelerometer is one of the most common inertial sensors in the world. Accelerometers can make inertial measurements of acceleration, vibration, and orientation (tilt) with one, two, or three axes.[1] The measurements are normally expressed in units relative to the standard acceleration of gravity (1 g = 9.8 m/s^2). When linear acceleration is zero, the only measured component is gravity. A 3-axis accelerometer at rest will always return an acceleration vector which is 1 g in magnitude. Figure 2.1 illustrates this with a 3-axis accelerometer at rest (left) and accelerating along the X-axis at a rate of 1 g (right).

As the sensor accelerates in a certain direction, a mass internal to the accelerometer is deflected from its base position relative to the sensor package. That deflection can be measured and used to compute the amount of acceleration that was required to produce it. Two common types of accelerometer are based on capacitive sensing and piezoelectric effect to convert the

[1] A 3-axis accelerometer is required to estimate tilt.

motion of the movable mass into an electrical signal. The following discussion explores the physics behind the capacitive model.

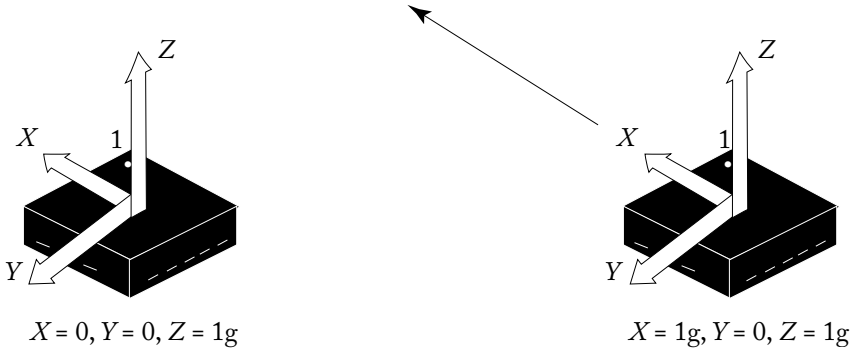

Figure 2.1: An example of accelation measurement.

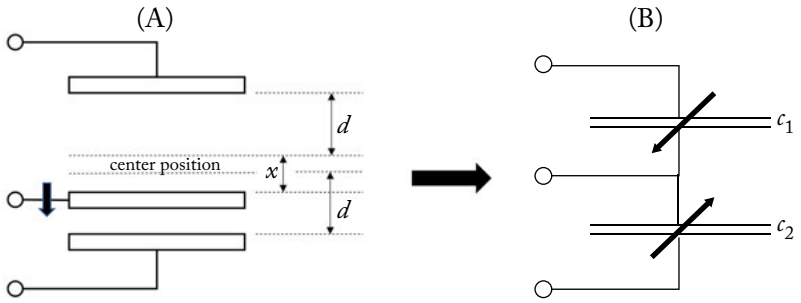

Figure 2.2: Single-axis transducer physical model (A) and equivalent circuit model (B). (Taken from the NXP MMA955xL Reference Manual, © 2017 NXP B.V.)

In the simplest case, a three-axis accelerometer consists of three surface capacitive sensing cells, also known as g-cells. The central mass in each g-cell moves relative to adjacent fixed beams, and its own at-rest position, when subjected to linear acceleration. This is shown for a single dimension in Fig. 2.2(A). When the central beam moves, the distance from the fixed beams on one side will increase by the same amount that the distance to the fixed beams on the other side decreases. The change in distance is a measure of acceleration. The g-cell beams form two back-to-back capacitors (Fig. 2.2(B)). As the center plate moves with acceleration, the distance between the beams change and each capacitor's value changes. The CMOS ASIC uses switched-capacitor techniques to measure the g-cell capacitors and extract the acceleration data from the difference between each set of two capacitors. The ASIC conditions and filters the signal, providing a high-level output voltage that is ratiometric and proportional to acceleration.

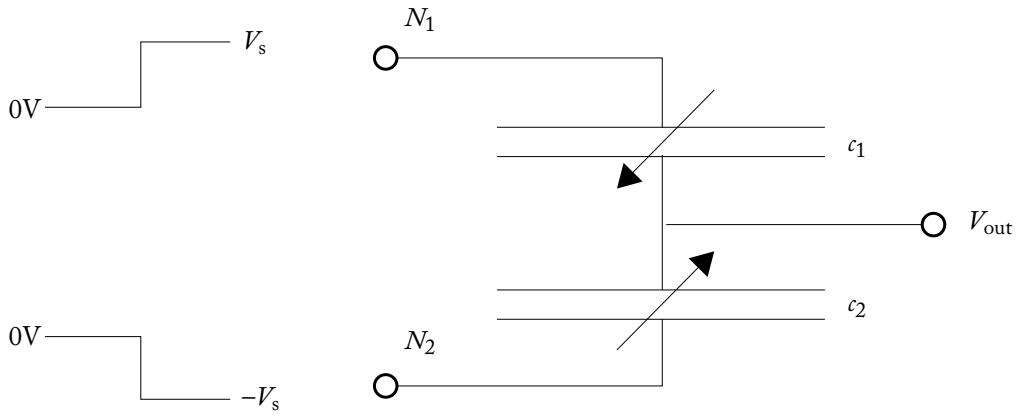

Figure 2.3: Dividers N_1 and N_2. (Taken from the NXP MMA955xL Reference Manual, © 2017 NXP B.V.)

Normally, the center mass in Fig. 2.2(A) is equidistant from both upper and lower plates such that c_1 and c_2 are equal. However, if the mass is displaced, the two capacitance values move in opposite directions. c_1 and c_2 form a capacitive divider. Figure 2.3 shows the outer legs of the divider (N_1 and N_2) driven with square waves that are 180 degrees out of phase from one another. When N_1 and N_2 switch from zero volts to V_s and V_{-s}, respectively, V_{out} is determined by the capacitor ratios as follows:

$$V_{out} = V_s \frac{c_1 - c_2}{c_1 + c_2}. \tag{2.1}$$

This can be converted to an expression of V_{out} as a function of displacement by considering capacitance theory. If $c_1 + c_2$ is modeled as standard, parallel-plate capacitors and ignore fringing capacitance, we begin by noting:

$$c_1 = \frac{n \cdot A \cdot \epsilon}{d + x} \tag{2.2}$$

$$c_2 = \frac{n \cdot A \cdot \epsilon}{d - x}, \tag{2.3}$$

where n is the number of beams in the g-cell (unit less), A is the area of the facing side of the beam in m^2, ϵ is the dielectric constant in farads per meter (1 farad = 1 coulomb per volt), d is the nominal distance between the beams in meters, x is the beam displacement in meters, and c_1 and c_2 are the measured capacitances.

Substituting Eqs. (2.2) and (2.3) into (2.1), we arrive at:

$$V_{out} = V_s \frac{c_1 - c_2}{c_1 + c_2} = -V_s \frac{x}{d}. \tag{2.4}$$

Next, we make use of Newton's second law:

$$F = M \cdot a, \tag{2.5}$$

where F denotes force applied to an object in Newtons ($1\ \text{N} = 1\ \text{kg} \cdot \text{m/s}^2$), M is the mass of the object measured in Kg, and a denotes the acceleration of the object measured in m/s^2, and Hooke's Law:

$$F = -k \cdot x, \tag{2.6}$$

where F is the displacement force in N, k is spring constant in N/m, and x is the beam displacement in meters. Combining Eqs. (2.5) and (2.6), we obtain

$$x = -\frac{M \cdot a}{k}. \tag{2.7}$$

Finally, substitute Eq. (2.7) into (2.4) to produce

$$V_{out} = V_s \frac{M \cdot a}{k \cdot d}. \tag{2.8}$$

Accelerometers are used in a wide range of applications. Many personal electronic devices such as cameras, smart phones, and tablets have accelerometers to track the orientation of the devices. When you rotate a smart phone from vertical to horizontal, the screen is automatically rotated as well. Accelerometers are also used in gaming [29], sports [3], and fitness [165, 166]. Freefall detection is also based on accelerometers, which is useful for detecting accidental falls of elderly people as well as computer devices [31]. Accelerometers are broadly used in vehicles [32]. They are used to improve suspension performance, estimate road condition, enhance vehicle active control, and trigger airbag systems. Machine condition monitoring is another common application for accelerometers. By embedding accelerometers in production equipment, the maintenance cycle of machinery can be optimized based on actual machine health [1, 2]. Predictive maintenance allows industries to save cost and avoid catastrophic failures because it can detect risky conditions before sudden failure occur.

2.1.1 ACCELEROMETER PLACEMENT

A question that should be asked during physical design of any product incorporating sensors is: "Where should sensors be located within the product?" Like almost anything else in life, "it depends …" is the answer. This section discusses accelerometer placement. If your application is such that the sensor is statistically fixed in space, or at least not rotating, it does not really matter where you mount the accelerometer on your sensor board. But if the device can rotate, things get a lot more interesting.

Assume that the accelerometer is located at point P in a portable consumer device as shown in Fig. 2.4. The device, which contains the accelerometer, is rotating in the rotating reference frame $\mathcal{B}$ as well as translating relative to the fixed reference frame $\mathcal{A}$. We want to obtain

the accelerometer measurement of the rotating reference frame $\mathcal{B}$ as viewed by an observer in the fixed reference frame $\mathcal{A}$. If the accelerometer P is located at the origin Q of the rotating frame (presumably the center of mass of the device), then the problem is simple. In the following, however, we will quantify the effect of the accelerometer placement (i.e., $\boldsymbol{r}$) using the rate of change transport theorem [33]. The transport theorem can be found in any rigid body dynamics textbook, and relates the rate of change of a vector $\boldsymbol{r}$ as observed in two different reference frames. The transport theorem is a fundamental relationship, based upon basic principles of geometry.

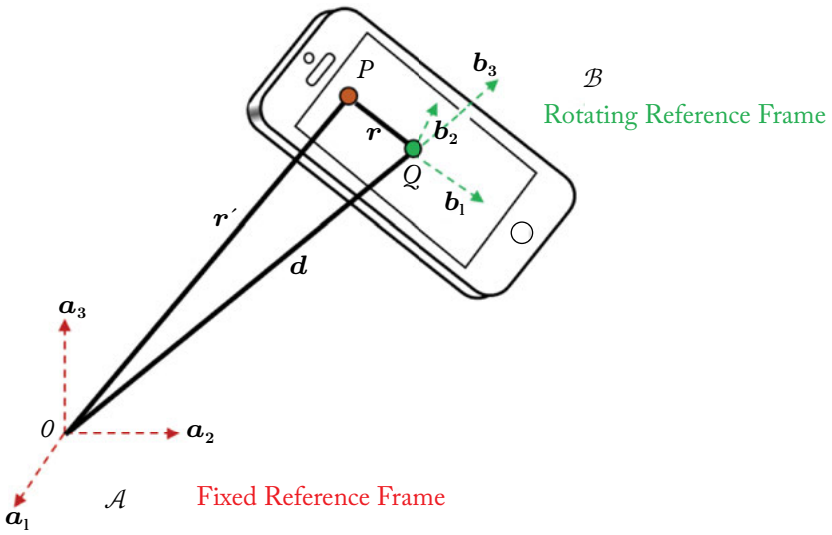

Figure 2.4: Accelerometer placement.

Let $\{\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3\}$ and $\{\boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_3\}$ be orthonormal bases fixed in $\mathcal{A}$ and $\mathcal{B}$, respectively. The vector $\boldsymbol{r}$ can be defined as

$$\boldsymbol{r} = r_1\boldsymbol{b}_1 + r_2\boldsymbol{b}_2 + r_3\boldsymbol{b}_3. \tag{2.9}$$

The rate of change of $\boldsymbol{r}$, as viewed by an observer in the frame $\mathcal{A}$, can then be written by

$$\frac{{}^{\mathcal{A}}d\boldsymbol{r}}{dt} = \frac{dr_1}{dt}\boldsymbol{b}_1 + \frac{dr_2}{dt}\boldsymbol{b}_2 + \frac{dr_3}{dt}\boldsymbol{b}_3 + r_1\frac{{}^{\mathcal{A}}d\boldsymbol{b}_1}{dt} + r_2\frac{{}^{\mathcal{A}}d\boldsymbol{b}_2}{dt} + r_3\frac{{}^{\mathcal{A}}d\boldsymbol{b}_3}{dt}, \tag{2.10}$$

where $\{r_1, r_2, r_3\}$ are scalars whose rate changes are not affected by relative motion of the reference frames, whereas $\{\boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_3\}$ are the vectors defined for the rotating reference frame. Therefore, $dr_1/dt, dr_2/dt$, and dr_3/dt are independent of the reference frame, whereas we need to explicitly denote ${}^{\mathcal{A}}d\boldsymbol{b}_1/dt, {}^{\mathcal{A}}d\boldsymbol{b}_2/dt$, and ${}^{\mathcal{A}}d\boldsymbol{b}_3/dt$ because they are taken with respect to the

reference frame $\mathcal{A}$. Equation (2.10) is rewritten as

$$\frac{{}^{\mathcal{A}}d\boldsymbol{r}}{dt} = \frac{{}^{\mathcal{B}}d\boldsymbol{r}}{dt} + r_1 \frac{{}^{\mathcal{A}}d\boldsymbol{b}_1}{dt} + r_2 \frac{{}^{\mathcal{A}}d\boldsymbol{b}_2}{dt} + r_3 \frac{{}^{\mathcal{A}}d\boldsymbol{b}_3}{dt}. \tag{2.11}$$

This simplifies to

$$\frac{{}^{\mathcal{A}}d\boldsymbol{r}}{dt} = \frac{{}^{\mathcal{B}}d\boldsymbol{r}}{dt} + {}^{\mathcal{A}}\boldsymbol{\omega}^{\mathcal{B}} \times \boldsymbol{r}, \tag{2.12}$$

where ${}^{\mathcal{A}}\boldsymbol{\omega}^{\mathcal{B}}$ is the angular velocity of the rotating reference frame $\mathcal{B}$ as viewed by an observer in the fixed reference frame $\mathcal{A}$. This expression is commonly referred to as the *transport theorem*. For more detailed derivation of Eq. (2.12), please refer to [33].

Let $\boldsymbol{r}' = \boldsymbol{r} + \boldsymbol{d}$. By repeatedly applying the transport theorem of Eq. (2.12) for $\boldsymbol{r}'$, we can convert the acceleration at the point P of the frame $\mathcal{B}$ into the one in $\mathcal{A}$. The resulting acceleration at P in frame $\mathcal{A}$ can be expressed as

$$\frac{{}^{\mathcal{A}}d^2\boldsymbol{r}'}{dt^2} = \frac{{}^{\mathcal{A}}d^2\boldsymbol{d}}{dt^2} + \frac{{}^{\mathcal{B}}d^2\boldsymbol{r}}{dt^2} + \left[2\left({}^{\mathcal{A}}\boldsymbol{\omega}^{\mathcal{B}} \times \frac{{}^{\mathcal{B}}d\boldsymbol{r}}{dt}\right) + \left(\frac{{}^{\mathcal{A}}d}{dt}{}^{\mathcal{A}}\boldsymbol{\omega}^{\mathcal{B}} \times \boldsymbol{r}\right) + {}^{\mathcal{A}}\boldsymbol{\omega}^{\mathcal{B}} \times \left({}^{\mathcal{A}}\boldsymbol{\omega}^{\mathcal{B}} \times \boldsymbol{r}\right)\right], \tag{2.13}$$

where the three last terms in the right-hand-side represent, respectively,

$$\text{(Coriolis Acceleration)} \qquad 2{}^{\mathcal{A}}\boldsymbol{\omega}^{\mathcal{B}} \times \frac{{}^{\mathcal{B}}d\boldsymbol{r}}{dt}, \tag{2.14}$$

$$\text{(Euler Acceleration)} \qquad \frac{{}^{\mathcal{A}}d}{dt}{}^{\mathcal{A}}\boldsymbol{\omega}^{\mathcal{B}} \times \boldsymbol{r}, \tag{2.15}$$

$$\text{(Centripetal Acceleration)} \qquad {}^{\mathcal{A}}\boldsymbol{\omega}^{\mathcal{B}} \times \left({}^{\mathcal{A}}\boldsymbol{\omega}^{\mathcal{B}} \times \boldsymbol{r}\right). \tag{2.16}$$

Basically, sensor fusion mathematics simplify dramatically when the accelerometer is at the center of mass (which is also the center of the rotating frame of reference). If you can make the assumption that, in your use case, the device is in a quasi-static state with no rotation, it doesn't matter. If not, you need to place the sensor at the center of mass or account for the effects mathematically. That involves keeping continuous track of the axis and rate of rotation and then making the adjustments.

2.2 MAGNETOMETER

We all are surrounded by the Earth's magnetic fields which are vector quantities characterized by their strength and direction. The commonly used units are tesla (T) and gauss (G):

$$1\ T = 10{,}000\ G. \tag{2.17}$$

Figure 2.5 [34] shows the Earth's magnetic field total intensity—the magnitude of field vectors. The geomagnetic field varies over the surface of the earth from approximately 25–65 $\mu\,T$. A *magnetometer* is used to measure magnetic field strength. You can subdivide magnetometers into vector magnetometers and total field (scalar) magnetometers. The former measures

the 3D vector components of a magnetic field, whereas the latter measures the magnitude of the vector magnetic field. Alternately, you separate magnetometers based on the physical properties used to make measurements. Examples include Hall effect sensors, current sensors, and magnetoresistive sensors. The proper choice of magnetometer type depends on the application. However, investigating all these variations is out of the scope of this book. We will focus on one recent technique—tunneling magnetic resistance (TMR), which is well suited for measuring the Earth's magnetic field.

TMR is based upon the magnetic tunnel junction (MTJ) illustrated in Fig. 2.6. The MTJ includes two magnetic layers separated by a thin insulating layer which allows tunneling when a voltage is applied. One of the two magnetic layers is called the "pinned (or, fixed) layer." It is permanently magnetized in one direction. The other magnetic layer is called the "unpinned (or, free) layer." As the magnetometer moves, changes in ambient magnetic fields will affect the unpinned layer. The tunneling current is maximized when the fields of these two layers are aligned in parallel, whereas the current is minimized when they are aligned in antiparallel. The tunneling current, therefore, affects the MTJ resistance change as given by

$$\frac{R_{ap} - R_p}{R_p} = 2\frac{P_1 P_2}{1 - P_1 P_2}, \tag{2.18}$$

where R_p and R_{ap} denote the resistances when the two magnetic layers are aligned in parallel and antiparallel, respectively, and P_1 and P_2 are the spin polarizations of the two magnetic layers.

Physically, each MTJ device can be optimized to sense magnetic fields in a specific direction. The magnetic sensor is formed by arranging arrays of these devices into three separate Wheatstone bridges as shown in Fig. 2.7 (each sensitive in a different direction) whose output voltage can be converted to digital format by an analog-to-digital converter.

Magnetometers are used in a variety of applications, including personal smart devices, industrial equipment, and healthcare. While the magnetometers are very useful in many applications, it is important to calibrate the magnetization measurement when the magnetic sensors are designed with soft- and hard-iron interferences.

2.2.1 HARD AND SOFT IRON MAGNETIC COMPENSATION

We have explained the basic operating principles for magnetometers and measuring magnetic fields. In the following, we will explore issues that you may encounter when using any magnetic sensor in consumer and industrial applications. To keep things simple, let us consider the case where you are integrating a magnetic sensor into a smart phone. Nominally, you would like to use your magnetic sensor to implement compass and navigation features. In a pristine environment, free from interference, we could take measurements directly from our sensor. The real world is not that simple.

Issue #1: Distortion of the magnetic field due to the presence of soft iron. Consider EMI shields, screws, and battery contacts. These generate no permanent magnetic field themselves.

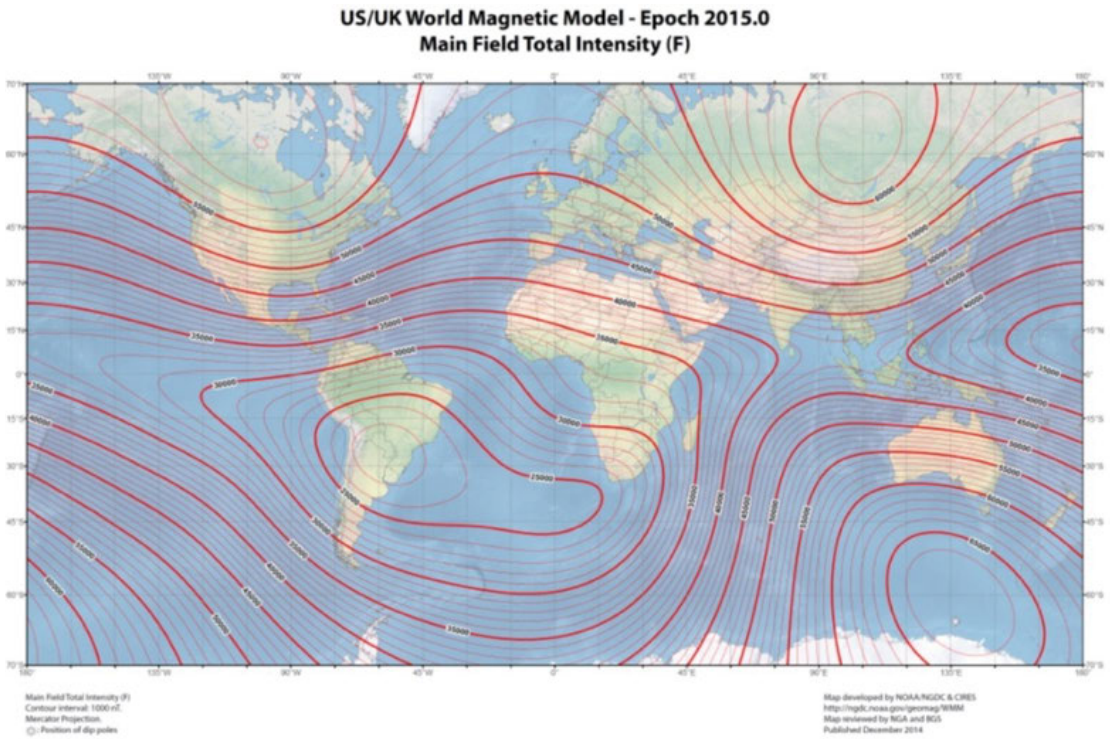

Figure 2.5: Earth's magnetic field intensity.[2]

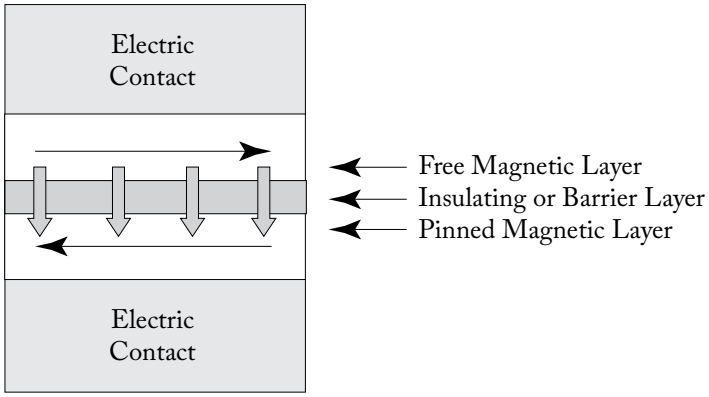

Figure 2.6: Magnetic tunnel junction (MTJ) technique (© 2017 NXP B.V.).

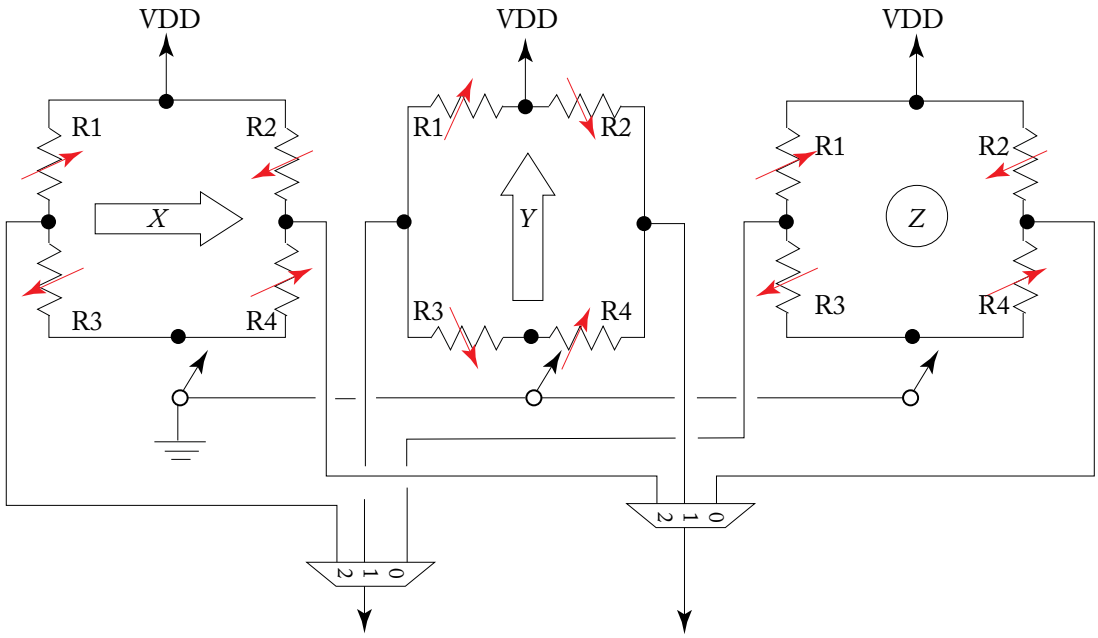

Figure 2.7: X, Y, and Z dimension MJTs arranged into separate Wheatstone bridges (© 2017 NXP B.V.).

But they do affect the ambient magnetic field by providing low impedance paths for magnetic flux. We call these types of materials "soft iron." To illustrate soft iron effects, a simulation with a U-shaped piece of steel sitting in a uniform magnetic field was performed. Figure 2.8A shows how the magnetic field, which would otherwise be shown as vertical lines, is distorted by the presence of the "U-bar." Steel provides a "lower resistance" path to the magnetic field than does the surrounding air. Thus, it is natural for the field to be diverted. Figure 2.8B takes that same U-bar and rotates it exactly 180 degrees in the same ambient field. You can see similarities in the field distortion. We can see just how similar "A" and "B" are by taking "B," and flipping it, first about one axis and then the other, to obtain Fig. 2.8C, which is identical in form to Fig. 2.8A. This makes a lot of sense when you realize that from the steel's perspective, "A" and "B" are identical except for the polarity of the ambient magnetic field. We should get symmetrical results. More importantly, we will be able to use this simple observation to remove the distortion caused by soft iron from our measurement of the ambient magnetic field.

To see how, consider this same U-bar and rotate it in 20 degree increments in the same field. At the same time, let's measure and plot the magnetic field at the "dot" you see nestled near the base of the "U." It's important to note that this point is fixed relative to the disturbing

[2]Map source https://www.ngdc.noaa.gov/geomag/WMM/data/WMM2015/WMM2015_F_MERC.pdf. Developed by NOAA/NGDC & CIRES.

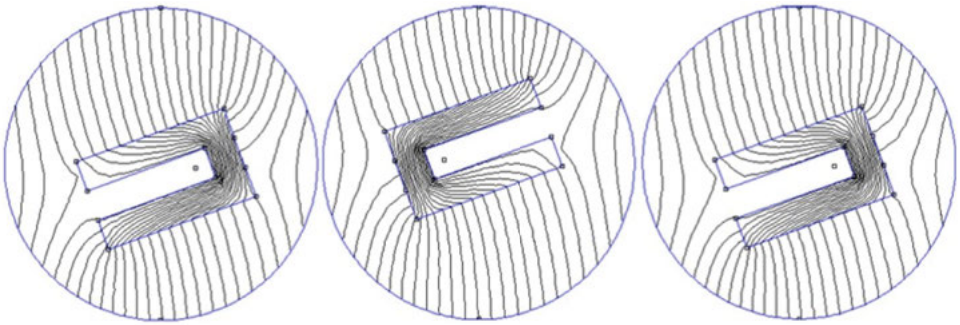

Figure 2.8: Soft iron distortion (© 2017 NXP B.V.).

metal. They rotate together. The symmetry shown in Fig. 2.9 continues to hold as we rotate our soft iron. The field distortion at each angle of rotation matches (after the "flips" noted above) the distortion seen at that angle +180 degrees. More importantly, the field magnitude measured at each angle matches the field magnitude measured at that angle +180 degrees. If we plot the x-y sensor readings for all our points, we will get an ellipse. This is a function of the basic physics, and always holds true, regardless of the sensor type used to make the measurement. If there were no soft iron present, and we simply rotated our sensor, the ellipse would collapse into a simple circle. Since the field remains the same regardless of the angle of measurement, this must be the case. So we see that the effect of soft iron is to distort a circle whose radius is equal to the magnitude of the ambient magnetic field into an ellipse.

This result can be extended to three dimensions. Measurements taken while rotating a sensor in free space undisturbed by hard or soft iron can be visualized as a sphere with fixed radius equal to the magnitude of the ambient magnetic field. Adding soft iron to the mix will distort that sphere into a 3D ellipsoid (Fig. 2.10).

The equation for a 3D ellipsoid in a general matrix form, if we have a column vector $\boldsymbol{x} = [x, y, z]^T$, is given by

$$(\boldsymbol{x} - \boldsymbol{\mu})^T \, \boldsymbol{\Sigma} \, (\boldsymbol{x} - \boldsymbol{\mu}) = 1, \tag{2.19}$$

where $\boldsymbol{\Sigma}$ is a 3×3 weight matrix and $\boldsymbol{\mu}$ denotes a 3×1 vector for the center of ellipsoid. If we take a representative set of samples on the surface of our ellipsoid, we can, through a variety of methods [35, 36, 37, 38], determine the reverse mapping from distorted to undistorted magnetic field readings. Essentially, we are curve fitting in three dimensions.

Issue #2: Distortion of the magnetic field due to the presence of hard iron "Hard iron" is another way of saying "permanent magnet." Most smartphones must have a speaker. And speakers have magnets. A lot of phone holsters have magnets to secure the device. So it turns out that we have to deal with them. The good news is that (compared to soft iron effects), compensating

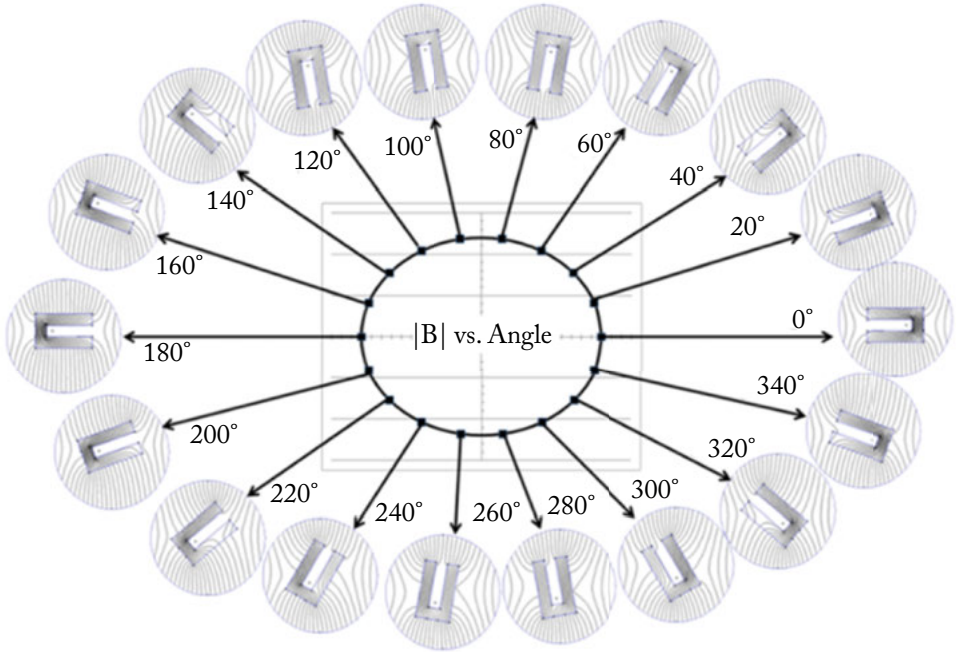

Figure 2.9: Soft iron (simulation with FEMM 4.2) [142] (© 2017 NXP B.V.).[3]

for hard iron offsets is relatively simple. If a magnet is fixed in location and orientation with respect to our sensor, then there is an additional constant field value added to the value that would otherwise be measured. If only soft iron effects are present, the ellipsoid of Eq. (2.19) should be centered at $\boldsymbol{\mu} = [0, 0, 0]^T$. A permanent magnet fixed relative to the measurement point simply adds an offset to the origin of the ellipsoid (Fig. 2.11).

If we have a large enough data set, e.g., N, we can determine the offset as

$$\text{Hard iron offset:} \qquad \boldsymbol{\mu} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n. \tag{2.20}$$

This technique will not work for magnets that move with respect to the sensor. The magnet on the phone holster flap cannot be permanently canceled out. But that is good news. A sudden shift in offset/magnitude of our calculated field probably implies that the phone has been inserted or removed from its holster. That can be a useful thing to know.

Implications The techniques discussed generally employ some form of curve fitting, which raises subtle questions that do not get discussed much: How many data points do we need in

[3]The FE simulations were done with the free FEMM 4.2 simulation tool [146] by Dr. David Meeker, who graciously offered examples and hints about the best way to run the simulations.

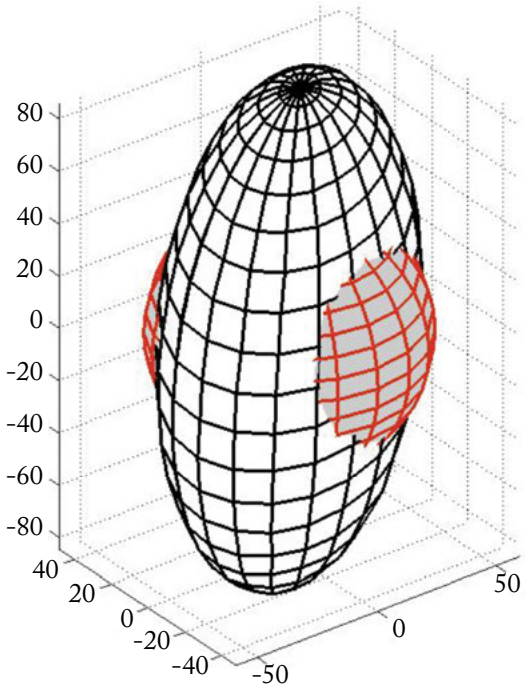

Figure 2.10: Ellipsoid distortion caused by soft iron effects.

our constellation of sample points? How often does that constellation need to be updated? How do we decide to add or drop points to/from the constellation? What should we do when a sudden change in ambient field magnitude is detected? What numerical method(s) should be used to calculate the trim parameters? How do you deal with uncorrelated magnetic disturbances that occur around us every day? How do you deal with field variations with temperature? The answers to these questions make up much of the "secret sauce" used by various vendors in their calibration algorithms.

2.2.2 MAGNETOMETER PLACEMENT

This section discusses tradeoffs to be considered when choosing where to locate a magnetometer in your design. The discussion builds on the previous sections, as well as material presented at Design West 2012 [39]. For the purposes of this discussion, we will consider the following use model: magnetometer as the magnetic component in an electronic compass. In this case, you desire to measure the earth's magnetic field, which ranges from 25–65 μT, depending upon your location upon the earth. This is a relatively weak field, and can easily be rendered unmeasurable unless you pay attention to magnetic issues during your system design. Many magnetic sensors

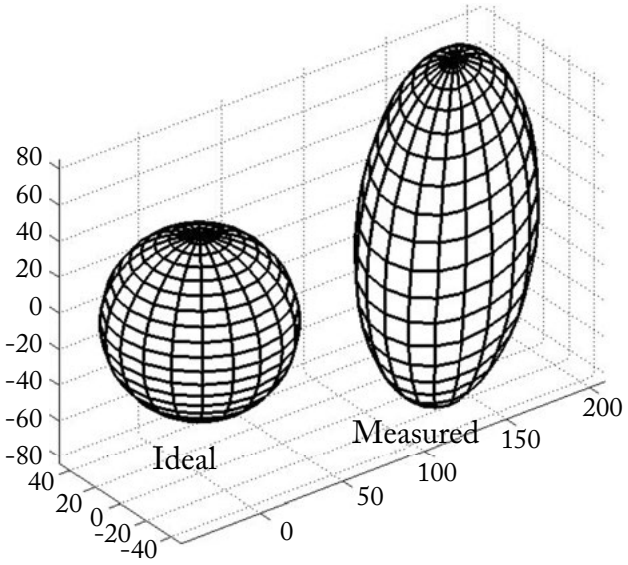

Figure 2.11: Effects of adding both soft and hard iron offset.

used for consumer and industrial applications have ranges much larger ($> 1000\ mT$) than the earth field, as they are optimized for sensing man-made fields.

In our context, soft iron includes ferrous EMI shielding, films for inductive chargers, screws, battery contacts, etc. Hard iron elements are permanent magnets within speakers, proximity switches, vibrators, etc. In addition, we may have magnetic fields resulting from speaker/-motor windings, PCB traces, etc.

If you do not have any of the above elements in your design, you can probably stop reading this section now. Unfortunately, most designs must deal with these issues. Let us start with general guidelines:

- Make the magnetometer placement the first step when outlining a new PCB layout.
- The ideal location for the magnetometer is for it to be as far away as possible from all sources of hard and soft interference and from high current traces.
- Time-varying current traces cannot be mathematically compensated for, whereas you may be able to treat constant current traces as hard iron interference, which can be compensated for.
- The best locations are often to be found on an edge or at a corner of the PCB.
- Place any Hall effect sensor and its magnet as far away as possible from the magnetometer.

- Other electromagnetic components may include loudspeakers, vibrators, hearing aid coils, batteries, and near field communication (NFC) antennas. These will create permanent hard iron, induced soft iron, and current sourced magnetic fields and should also be placed as far as possible from the magnetometer.
- Do not attempt to shield the magnetometer from magnetic fields generated by components on the PCB. Remember, our goal is to measure the earth's magnetic field. If you shield the sensor magnetically, you will destroy the very signal you are trying to measure.
- If shield cans are required to isolate sensitive subsystems such as radios from each other, then they must be made from low permeability conductive materials.[4]
- Industrial design and mechanical engineering teams should be involved at the start of any project involving a magnetometer. All mechanical components should be reviewed for their ferromagnetic content and replaced with low permeability alternatives.

Although we can mathematically compensate for many soft iron impacts, it is still best to avoid them when we can. As an example, we've simulated the effects of a small piece of steel on the value of the magnetic field measured nearby. Assume 1" by 1" by 0.02" thick steel as shown in Fig. 2.12A. If you were to look at that piece of steel from the side, you have the view shown in Fig. 2.12B. Figure 2.12C is the same edge view, but now we have added some information to specify simulation conditions.

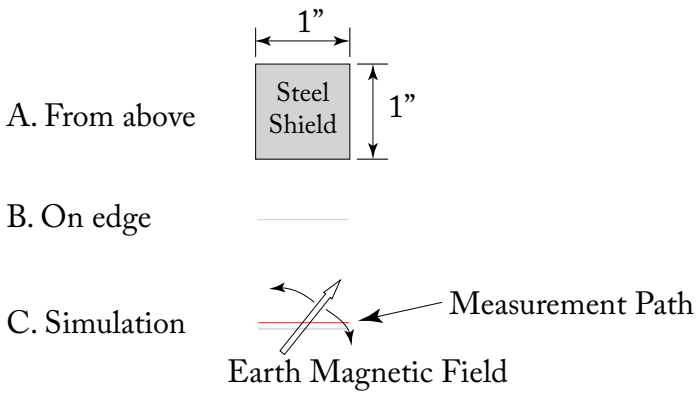

Figure 2.12: 1 in x 1 in x 0.02 in steel shield (© 2017 NXP B.V.).

For our simulation, it is assumed that an earth's magnetic field is approximately 40 μT, which is free to rotate over a range of 0 to 180 degrees during the simulation. This is equivalent

[4]Such as brass, beryllium-copper alloys, aluminum, nickel-silver alloys (such as 752 or 754), or even low permeability stainless steel alloys (such as SUS305 or SUS316). The aim is to combine low electrical resistance (to minimize electrical field leakage) with low magnetic permeability (to minimize induced soft iron fields). It is good practice to keep even low permeability shields a minimum of 7 mm away from the magnetometer sensor.

to holding the field static, and rotating the sensor (which is what happens in the real world). The red line in Fig. 2.12C represents a range of points in which we will measure the magnetic field intensity for each of those variations in field direction. The results were extracted and summarized via Excel, and are shown in Fig. 2.13.

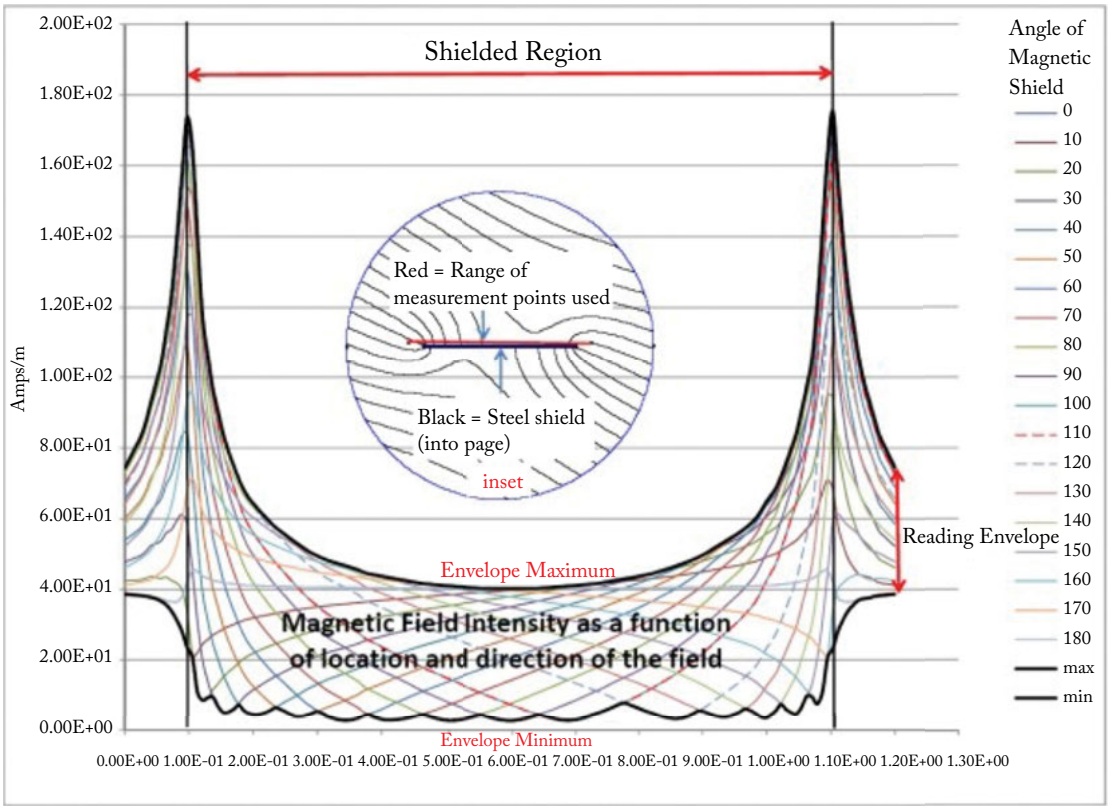

Figure 2.13: 2D finite element simulation results (© 2017 NXP B.V.).

The results are pretty much what you expect. Soft iron effects distort the magnetic field as a function of the angle between the ferrous material and the magnetic field. This results in an "envelope" of when you plot the magnitude of possible magnetic field values.

Significant field attenuation is possible either immediately above or below any magnetic shield. Do not put your sensor there! The worst place to put your sensor is immediately above or below the center of a magnetic shield. This is easy to do if you are not paying attention. Shielding used for EMI may also be magnetic, or you may be including a film used for inductive charging in your design. Both can really screw up your chances for getting consistent magnetic readings.

Field lines tend to converge near the edges of ferrous materials. You see that here with the wide spread between max and min values in the envelope near the edge of the shield. If

you go back to Section 2.2.1 and review our prior discussion on the elliptical distortion effects of soft iron on sensor readings, you will see that this spread of max and min field readings directly translates in more distortion being seen in the 3D ellipsoid of sensor readings. Magnetic compensation software can correct for much of this distortion, but as the ellipsoid becomes extreme, this correction will become less effective.

Finally, even a little distance between your sensor and the shield helps tremendously. You can see that the min/max ratio drops to less than 2 only 0.1 in from the shield on both sides. This is fairly modest distortion, and should be easily correctable.

Next, consider impacts on magnetic field resulting from wires and traces on your PCB. We can use the Biot–Savart law[5] to estimate the effect of wire and trace currents on sensor readings. For long wires and traces, Biot–Savart can be simplified to:

$$|\boldsymbol{B}| = \frac{\mu_0 I}{2\pi r}, \tag{2.21}$$

where $\boldsymbol{B}$ denotes magnetic field in T, $\mu_0 = 4\pi 10^{-7} T \cdot m/A$, I is current in amps (A), and r is distance from wire/trace to sensor in meters (m). This can be rearranged to:

$$I \leq 5r\,|\boldsymbol{B}|, \tag{2.22}$$

where $\boldsymbol{B}$ is now in μT. If you know the field magnitude you can afford to ignore and the distance from your sensor to trace/wire, then you can use Eq. (2.22) to calculate the maximum wire/trace current. As an example, let:

$$\begin{aligned} |\boldsymbol{B}| &= 0.1\ \mu T \\ r &= 10 \text{ mm distance from sensor to wire/trace.} \end{aligned}$$

Then, the maximum current allowed in the trace would be

$$I \leq 5\,r\ |\boldsymbol{B}| = 5 \times 10 \times 0.1 = 5\,mA. \tag{2.23}$$

We can treat the DC component of this current as a hard iron offset (it looks like a magnet). AC components show up as "noise" in our sensor output.

It is obvious that adding a magnetic sensor to your design should be done early in the design cycle with careful consideration with respect to surrounding elements in the physical design.

2.3 GYRO SENSOR

Gyro sensors measure angular rate of rotation in one, two or three dimensions [40]. Gyro sensors are used for detection, stabilization, and control of rotational motion. Figure 2.14 illustrates the three rotations—roll, pitch, and yaw for the Aerospace frame of reference. This is also known

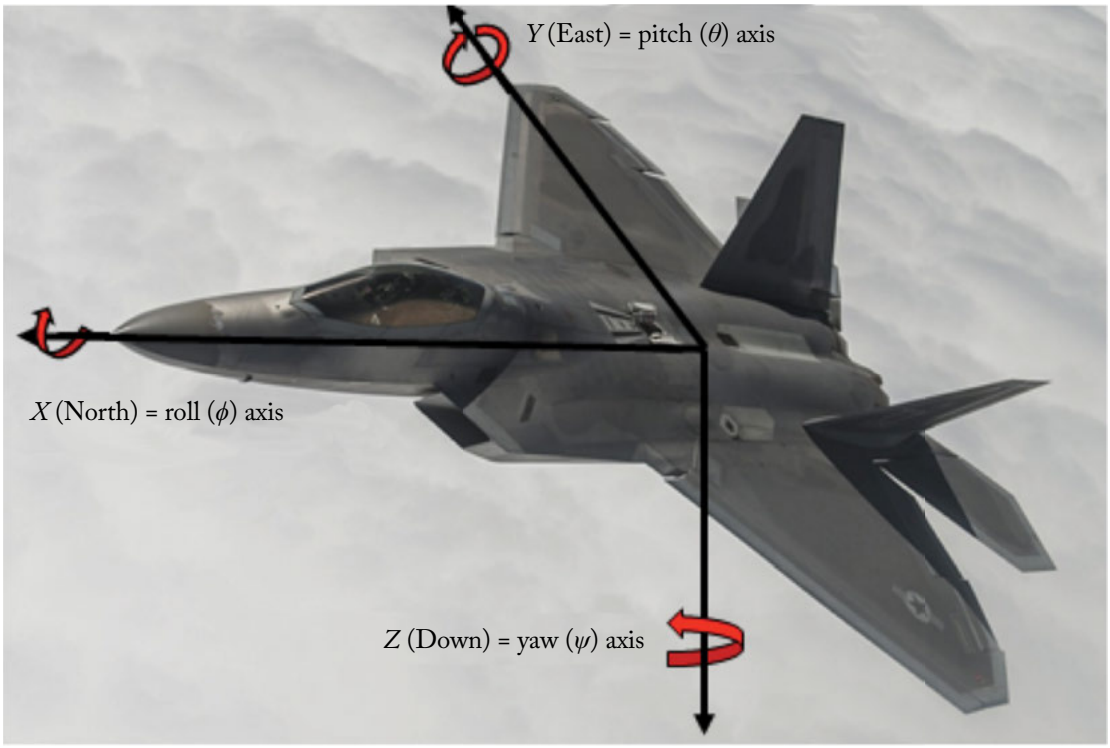

Figure 2.14: 3-axis rotations in the aerospace frame of reference.

as the NED frame of reference, as the X, Y, and Z axes are aligned with the North, East, and Up directions.

The use of MEMS technology allows the miniaturized design of gyro sensors by using tiny vibrating structure in the sensor. The operating principle of gyro sensors relies on the Coriolis effect.[6]

In Fig. 2.15 the sensor frame is attached to the sensor package. Drive mass M_D is electrostatically stimulated to vibrate in direction $\boldsymbol{v}$ (horizontal relative to the sensor frame) by alternating voltage V_{Drive} across capacitor C_D. M_D is constrained to move only in the horizontal direction relative to the sensor frame, to which it is attached by springs S1 and S2. Sense mass M_s moves horizontally *with* the drive mass, and is constrained to move only vertically *with respect* to M_D.

When rotation along the z-axis (yaw) is applied, M_S sees a force with respect to M_D, that is defined by

$$f_{Coriolis} = -2\, m\, (\boldsymbol{v} \times \boldsymbol{\omega}), \tag{2.24}$$

[5] https://en.wikipedia.org/wiki/Biot%E2%80%93Savart_law
[6] https://en.wikipedia.org/wiki/Coriolis_force

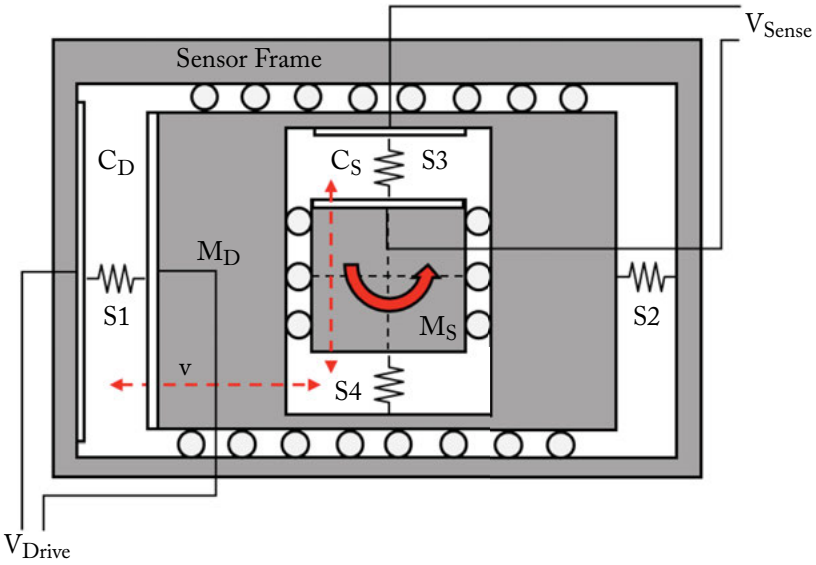

Figure 2.15: Single-axis MEMS gyro with separate drive and sense masses.

where m is the mass of M_S, vibrating in direction $\boldsymbol{v}$, with the angular rotation rate $\boldsymbol{\omega}$ in yaw. This changes the spacing of the plates in capacitor C_S, which in turn causes a change in capacitance that is proportional to the value of angular rotation rate $\boldsymbol{\omega}$. The capacitance value is converted to an output voltage and then converted again to digital values for the gyro sensor.

There are many types of gyros. The example above is but one variant chosen to illustrate how a MEMS structure can utilize the Coriolis force to measure rotational rate. MEMS gyros are used in applications including digital cameras, vehicles, game controller, smart phones and tablets, navigation, and gesture-based user interfaces.

2.4 PRESSURE SENSOR/ALTIMETERS

MEMS pressure sensors convert pressure into an electric signal whose magnitude varies proportionally to pressure. There are many types of pressure sensors, here we consider Piezo-Resistive pressure transducers (PRTs).

PRTs measure pressure via a Wheatstone bridge in which four pressure-sensitive resistors are located on a thin MEMS membrane which has a pressure gradient across it. The orientation of the resistors is such that two increase and two decrease in value with increasing pressure. The resistors are also temperature dependent, and modern devices include circuitry to compensate for temperature as well as nonlinear pressure effects. This yields a sensor in which pressure ($\boldsymbol{p}$)

can be defined as the force ($\boldsymbol{f}$) per unit area (A):

$$\boldsymbol{p} = \frac{\boldsymbol{f}}{A}, \tag{2.25}$$

where $\boldsymbol{f}$ is perpendicular to the sensor surface.

Atmospheric pressure can be used to estimate the altitude of an object above a reference level—usually sea level. The NOAA, NASA, and the USAF *U.S. Standard Atmosphere, 1976* contains expressions for "an idealized, steady-state representation of the earth's atmosphere from the surface to 1,000 km." The standardized atmospheric pressure can be described as a function of altitude:

$$\text{Altitude in meters} = K_1 \left(1 - \left(\frac{p}{p_0}\right)^{K_2}\right), \tag{2.26}$$

where $K_1 = 44,220.77$ meters, $K_2 = 0.190263$, and $p_0 = 101,325$ Pascals (at sea level). This nonlinear function is shown in Fig. 2.16. Note that our everyday lives are almost always in the range of 50–110 kPa. A few examples of extreme cases of altitudes on the earth include:

- Mount Everest, which is the highest, is 8,848 m high;
- the Dead Sea at −422 m below "sea level;"
- the highest town in the world is Wenzhuan, which is 5,099.3 m above sea level; and
- the highest town in the U.S. is Leadville, CO, at 3,179.1 m above sea level.

Pressure altimeters are widely used in our everyday activities such as hiking, outdoor/indoor localization for GPS assist or location-based services, and in aviation control systems.

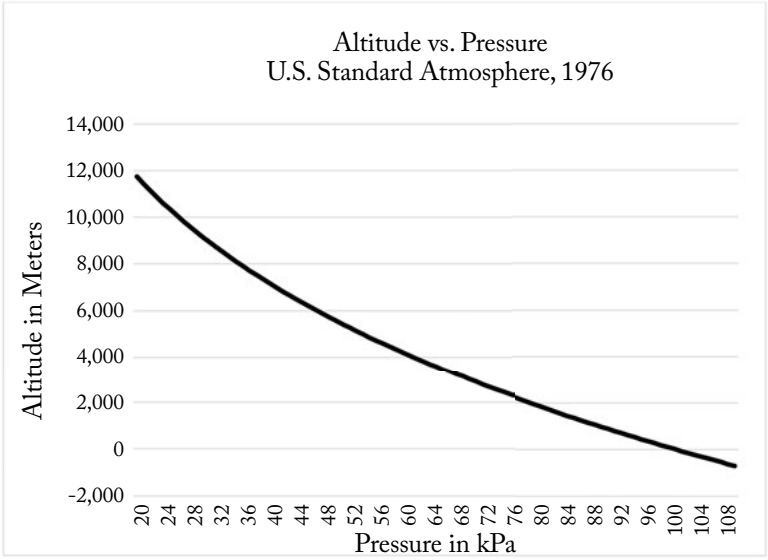

Figure 2.16: Altitude vs. atmospheric pressure.

CHAPTER 3

Sensor Fusion

Sensor fusion is a process by which data from different sensors are combined such that the resulting information is more accurate, more complete, or more dependable than could be determined by any single sensor alone. Sensor fusion results in synergistic effects that can be decomposed into a combination of four fundamental aspects: *redundancy*, *complementarity*, *timeliness*, and *cost* [30, 41, 42, 43]. Redundant information may be obtained by using multiple sensors to observe the same set of features. The fusion of redundant information can reduce noise and uncertainty in sensor data. Complementary information from multiple sensors can improve reliability of information and even generate new features that could not be achieved by individual sensors. Operating multiple low-precision sensors in parallel may reduce measurement time compared to the overall speed of a single, higher precision, sensor. Lastly, sensor fusion can generate additional functions which may be expensive or even impossible for a single sensor to measure.

Recent research in autonomous cars is driving advanced applications which fuse inertial, GPS, vision, radar, lidar, and other sensors to generate a 3D representation of the area about a vehicle. This book focuses on simpler "inertial sensor fusion" examples, but some of the techniques (such as Kalman filtering) are common to even the more complex applications.

We can easily see inertial sensor fusion techniques in use in the great variety of iPhone and Android sensor applications. One such applications is the *3D Compass* application. This is an application that fuses magnetometer, accelerometer, and GPS and video information to display not only the location of the user, but also the user's perspective. The application screen, as shown in Fig. 3.1, provides a current camera view overlaid with a virtual compass and a map oriented in the same way the user is facing.

The operating principle of the sensor fusion application in Fig. 3.1 is to estimate and track the *orientation* relative to three reference axes. This can be done by fusing the outputs of a three-axis accelerometer with a three-axis magnetometer. The accelerometer provides the earth gravity vector which is perpendicular to the ground, while the magnetometer works as a compass providing the geomagnetic vector. By combining these complementary sensors, we can compute a new feature—3D orientation. Although the measurements from an accelerometer and a magnetometer may be enough to compute the 3D orientation, the estimated output can be affected by unexpected disturbance such as the hard- or soft-iron interference and random noise. Additionally, we may consider the *redundancy* aspects of sensor fusion by adding multiple sets of accelerometer and magnetometer. This allows us to increase the accuracy of the orientation estimation, but may not be practical due to the number of sensors required. Another aspect

Figure 3.1: Augmented reality (AR) utilizing sensor fusion (© 2017 NXP B.V.).

we can think of is *complementarity*. A gyroscope itself can provide orientation estimation relative to an initial orientation (which can be provided by the accelerometer/magnetometer pair). The output of gyroscope is angular rate of rotation about the three axes. Integrating the angular rotation rates over a time period yields angular rotation increments. After summing all the increments, therefore, we can obtain the orientation estimation relative to the starting orientation. But if there was a small error during the integration, the small error is accumulated over time and responsible for *drift* in the resulting orientation estimation. While the accelerometer and magnetometer pair is weak for external disturbance such as iron interference and random noise, the gyroscope is robust against it. On the other hand, while the gyroscope causes drift when orientation is estimated, the accelerometer and magnetometer pair is not affected. Combining the two schemes can provide better estimation of orientation.

The orientation estimation has been widely used for a number of applications including: human body segment orientation [44, 45, 46, 47, 48], human motion analysis [49, 50], robotics [51, 52], aerospace [53], navigation [54, 55], machine interaction [56], and so on.

There are many diverse types of sensor fusion applications. The orientation estimation explained above is just one of them. Since the gyroscope is relatively expensive compared to other sensors, and also consumes much power than the others, one can develop a virtual gyroscope by combining an accelerometer and a magnetometer. We will describe how to fuse those two sensors for a gyroscope in Section 3.5.2 (page 61). Table 3.1 describes more applications of sensor fusion techniques with typical minimum sensor complements.

In summary, we can state that sensor fusion encompasses a variety of techniques which can:

Table 3.1: Typical minimum sensor complements for sensor fusion applications

Applications	Accelerometer	Magnetometer	Gyroscope	Pressure Sensor
eCompass, Pointing/remote control, Augmented/virtual reality	X	X		
Virtual gyroscope	X	X		
Gyro-compensated eCompass	X	X	X	
Activity monitors	X	X		
	X		X	
Motion capture	X	X	X	
3D mapping and localization	X	X	X	X
Image stabilization, Gesture recognition	X		X	

- Trade off strengths and weaknesses of the various sensors to compute something more than can be calculated using the individual components; and
- improve the quality and noise level of computed results by taking advantage of:
  - known data redundancies between complement sensors and
  - knowledge of system transfer functions, dynamics and/or kinematics.

3.1 TERMINOLGY

Before proceeding further with our inertial sensor fusion discussion, it is useful to define our terminology.

3.1.1 DEGREES OF FREEDOM (DOF)

Any movement of a rigid body can be characterized as a translation in space plus an optional rotation about the body origin. The first equates to distances $\Delta x, \Delta y, \Delta z$ along the three global Cartesian axes, respectively. The second equates to rotations ϕ, θ, ψ about the local body frame axes (see Fig. 2.14). Motion therefore requires six numbers or *degrees of freedom* (plus time) to fully specify the motion.

Consider the rigid body trajectory from A to D as shown in Fig. 3.3. The movement can be characterized by translation as the body moves along the points A, B, C, D. At the same time, the rigid body also rotates to various orientations which can be described via Euler angles

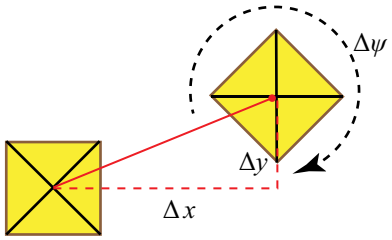

Figure 3.2: Simple rotation and translation.

ϕ, θ, and ψ. In Fig. 3.3, the blue line shows the *XYZ* locations over time. The cyan lines off the trajectory line specify the orientations along the way.

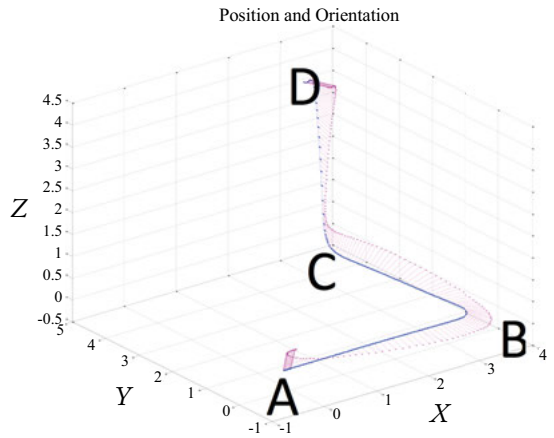

Figure 3.3: Translation plus orientation as vectors.

3.1.2 AXIS/AXES

Accelerometers and magnetometers each measure three-dimensional vector quantities. Acceleration plus gravity in the first case, and magnetic field intensity in the second. Measurements are performed relative to a standard Cartesian coordinate system. Gyroscopes measure motion in terms of angular rate *about* the axes of the same coordinate system. To measure the full quantity, we require that each sensor have three measurement *axes*.

As we have seen, pressure sensors measure a single scalar quantify. We refer to these as a single-axis device.

To simplify the mathematics, it is helpful if all 3-axis devices utilize the same Cartesian coordinate system. They should also obey the Right-Hand-Rule (RHR) conventions for axis alignment and direction of rotation as shown in Fig. 3.4 and Fig. 3.5, respectively.

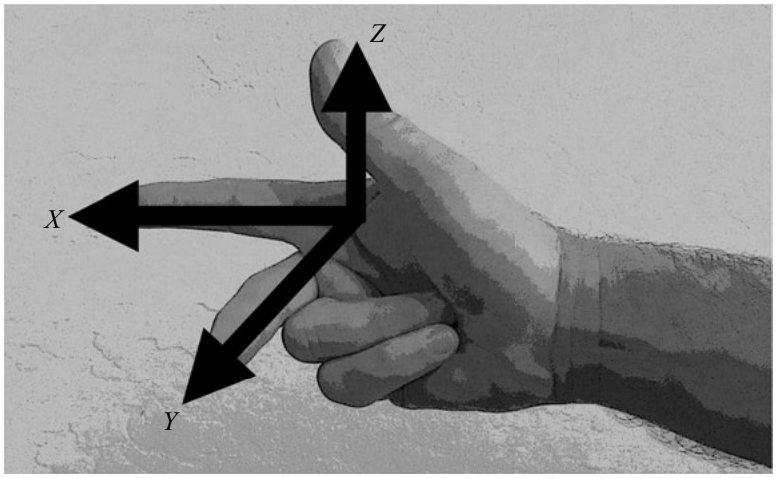

Figure 3.4: Right hand coordinate system axes alignment.

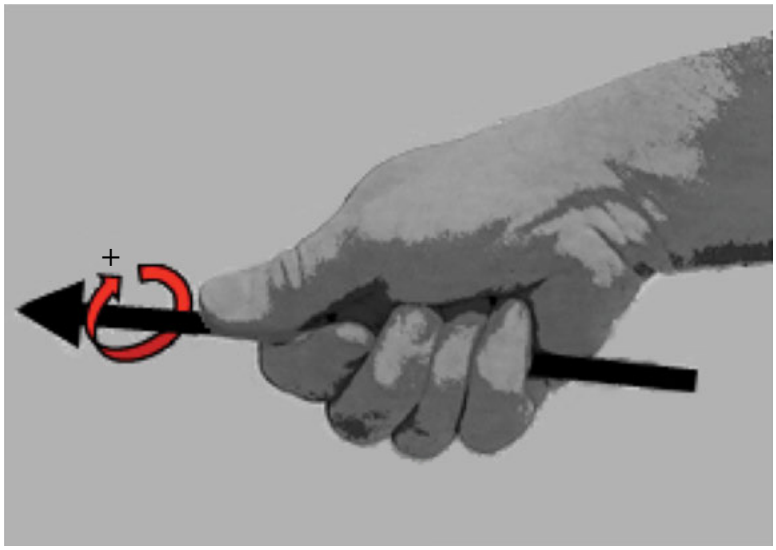

Figure 3.5: Right hand coordinate system rotation.

3.1.3 SENSOR MODULE CONFIGURATIONS

You should use *DOF* when describing motion and use *axis* (or *axes*) when describing sensor configurations. The following paragraphs elaborate on quantities measured by our motion sensor systems, and define standard terms for various sensor configurations.

A basic 3-axis accelerometer returns values for linear acceleration in each of three orthogonal directions. Figure 3.6 illustrates the orthogonal directions of the axes.

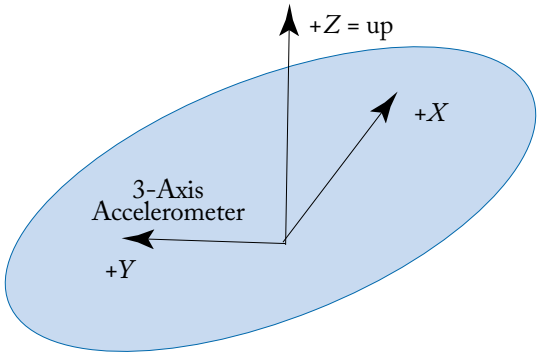

Figure 3.6: A 3-axis accelerometer returns X, Y, and Z acceleration in the sensor's frame of reference.

Gyroscopes return rates of rotation about each of the three sensor axes, as illustrated in Fig. 3.7. Integrating the rates of rotation results in the angles ϕ, θ, and ψ correspond to changes in orientation.

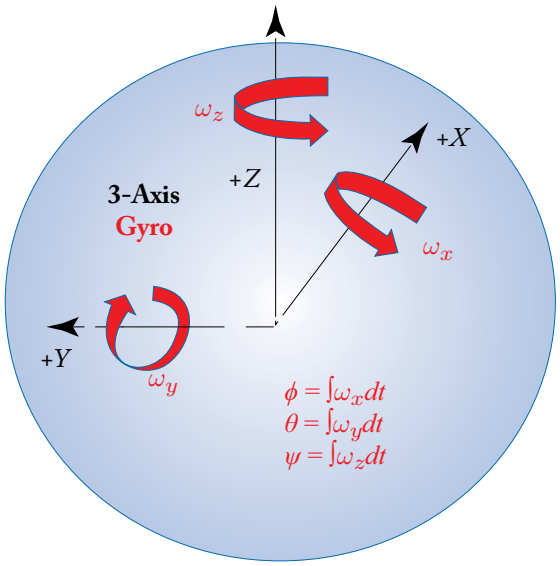

Figure 3.7: A 3-axis gyro returns rotation rates about each of X, Y, and Z axes.

A 3-axis magnetometer (Fig. 3.8) will return x, y, z components of the ambient magnetic field. This is nominally the earth field for many applications, but may include significant offsets and distortions due to hard- and soft-iron interferences.

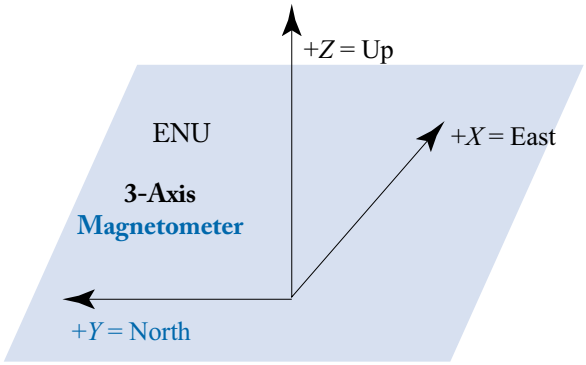

Figure 3.8: A 3-axis magnetometer will allow you to align yourself with the earth's magnetic field.

Accelerometer, gyroscope, and magnetometer each returns a 3-dimensional vector. The pressure sensor returns a single scalar value, as illustrated in Fig. 3.9. Pressure can be used to infer changes in altitude, which adds another source of information when computing vertical locations.

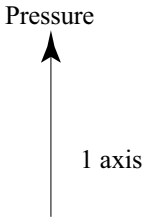

Figure 3.9: Pressure is our 10th axis.

Now we define commonly used sensor configurations. By combining an accelerometer with a gyro, we obtain a 6-axis inertial measurement unit (IMU) as illustrated in Fig. 3.10. In addition to an IMU, we obtain 9-axis sensor unit. When a magnetometer is added, as shown in Fig. 3.11, it is called a MARG (Magnetic, Angular Rate, and Gravity) sensor. If a computing engine is added to a MARG, then we get an AHRS (attitude and heading reference system). Sometimes we may want to add a pressure sensor to MARG or AHRS, then we get a slightly smarter MARG or AHRS. We can refer to it as a 10-axis solution (Fig. 3.12).

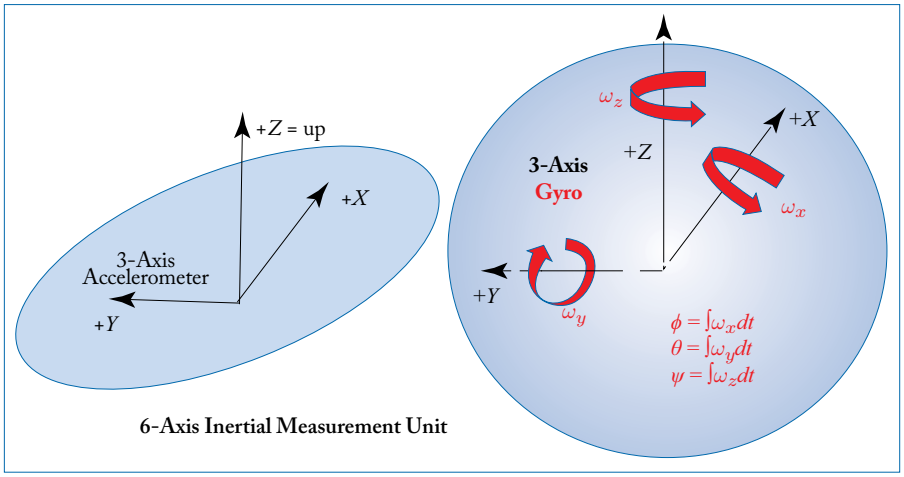

Figure 3.10: 6-axis IMU = gyro + accelerometer.

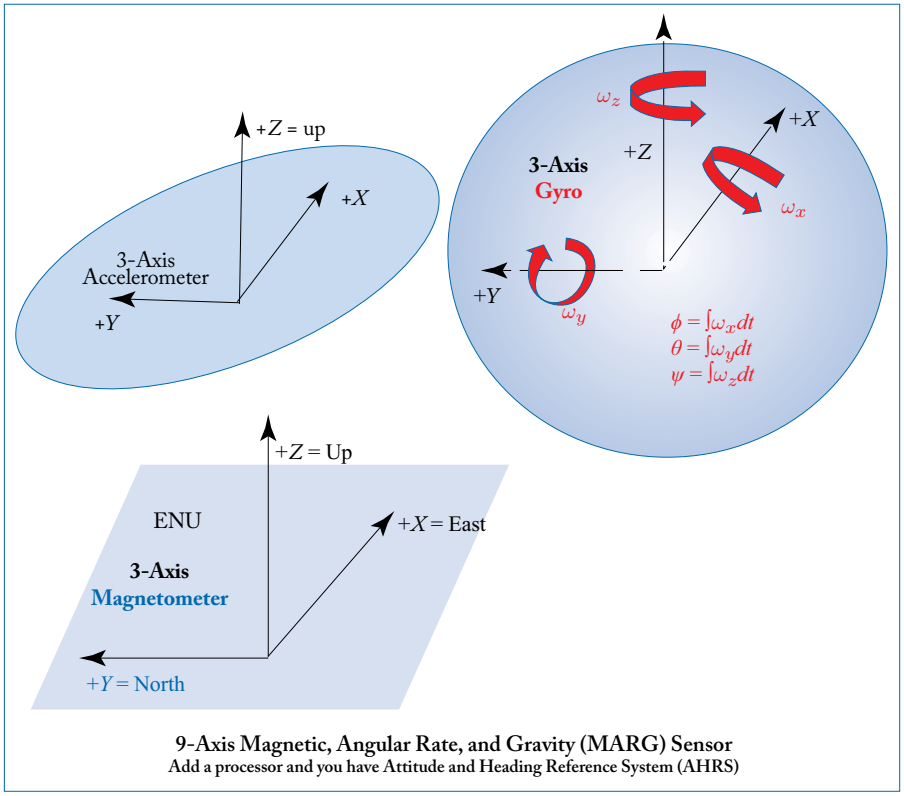

Figure 3.11: 9-axis MARG.

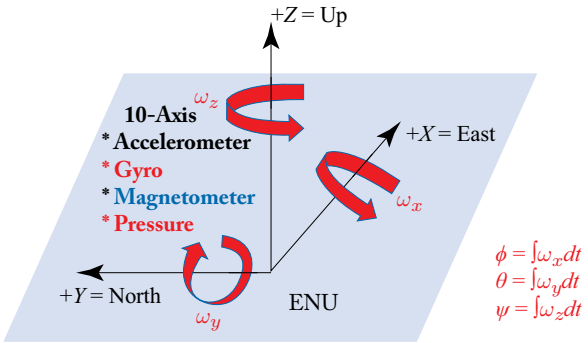

Figure 3.12: A full 10-axis sensor subsystem = accelerometer + gyro + magnetometer + pressure.

3.2 BASIC QUATERNION MATH

A strong understanding of basic geometry and orientation representations is required to work on inertial sensor fusion problems. The quaternion[1] is arguably the most efficient representation for orientation, and will be discussed in that context in Section 3.3. But first we explore the foundations of quaternion math here.

3.2.1 INTRODUCTION AND BASIC PROPERTIES

Conventionally, complex numbers have been widely used for geometric interpretation of two-dimensional vectors. Consider a complex number $x + iy$ that corresponds to a vector $\boldsymbol{a} = [xy]^T$ in a coordinate plane. The complex number component satisfies $i^2 = -1$. The magnitude of the vector $\boldsymbol{a}$ is given by $|\boldsymbol{a}| = \sqrt{x^2 + y^2}$ and an angle θ from $\tan\theta = y/x$. It is easy to see that addition of complex numbers is equal to the addition of vectors. When complex numbers are multiplied, it turns out that the magnitude of the product results in the product of the magnitudes of the factors and the angle of the product is equal to the addition of the angles of the two factors. Let $|\boldsymbol{a}| = r$. We learn from high school trigonometry of a triangle that is $x = r\cos\theta$, $y = r\sin\theta$, and $a = x + iy = re^{i\theta}$. Then the product of two complex numbers a_1 and a_2 results in $a_1 a_2 = r_1 r_2 e^{i(\theta_1+\theta_2)}$ where $r_1 r_2$ is the product of magnitudes and $\theta_1 + \theta_2$ is the summation of the angles. For a unit vector, i.e., $r_1 = r_2 = 1$, the complex number multiplication is a useful baseline to provide a rotation operator in a *plane*.

Although the complex numbers for the polar representation is simple and useful in 2D space, the geometric interpretation using complex numbers in 3D space is not easy. Quaternions are a type of hyper-complex numbers [57] that use triplets of complex number components. Instead of single complex number component i, it requires three components satisfying the

[1]Some properties of quaternions are introduced here. For more details, refer to [57].

relationship:

$$\begin{aligned} \boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = \boldsymbol{ijk} &= -1 \\ \boldsymbol{ij} &= \boldsymbol{k} = -\boldsymbol{ji} \\ \boldsymbol{jk} &= \boldsymbol{i} = -\boldsymbol{kj} \\ \boldsymbol{ki} &= \boldsymbol{j} = -\boldsymbol{jk}, \end{aligned} \tag{3.1}$$

where $\boldsymbol{i}$, $\boldsymbol{j}$, and $\boldsymbol{k}$ are the standard orthonormal bases in three-dimensional space $\mathbb{R}^3$. The vectors may be written as triplets of real numbers:

$$\begin{aligned} \boldsymbol{i} &= [1\ 0\ 0]^T \\ \boldsymbol{j} &= [0\ 1\ 0]^T \\ \boldsymbol{k} &= [0\ 0\ 1]^T. \end{aligned} \tag{3.2}$$

Just as a complex number $x + iy$ is used to describe rotation in a plane, quaternions provide an efficient operator to describe rotations in 3D space. In this section, we briefly review quaternion algebra before describing the quaternion based rotation operator for 3D space in the next section.

A quaternion q is a four-tuple, defined as

$$q = (q_0, q_1, q_2, q_3), \tag{3.3}$$

where the components q_0, q_1, q_2, and q_3 are real numbers. We can write a quaternion as a form of hyper-complex number where, instead of scalar values of real and imaginary components, we have a real scalar and a vector imaginary component. A quaternion q can be defined by a scalar part q_0 and a vector part $\boldsymbol{q}$ in $\mathbb{R}^3$:

$$q = q_0 + \boldsymbol{q} = q_0 + \boldsymbol{i}\, q_1 + \boldsymbol{j}\, q_2 + \boldsymbol{k} q_3, \tag{3.4}$$

where $\boldsymbol{i}$, $\boldsymbol{j}$, $\boldsymbol{k}$ are the orthonormal bases defined in Eq. (3.2). The notations of quaternions seem strange: the sum of a scalar and a vector, which is not ordinarily defined in conventional linear algebra. We introduce some quaternion definitions and properties in what follows. For those who are interested in more details, please refer to [57].

3.2.2 EQUALITY

Let p and q denote quaternions:

$$\begin{aligned} p &= p_0 + \boldsymbol{i}\, p_1 + \boldsymbol{j}\, p_2 + \boldsymbol{k}\, p_3, \\ q &= q_0 + \boldsymbol{i}\, q_1 + \boldsymbol{j}\, q_2 + \boldsymbol{k} q_3. \end{aligned} \tag{3.5}$$

Two quaternions p and q are equal if and only if $p_0 = q_0$, $p_1 = q_1$, $p_2 = q_2$, $p_3 = q_3$.

3.2.3 ADDITION

The *sum* of two quaternions p and q is defined by adding the corresponding components:

$$p + q = (p_0 + q_0) + \boldsymbol{i}\,(p_1 + q_1) + \boldsymbol{j}\,(p_2 + q_2) + \boldsymbol{k}\,(p_3 + q_3)\,. \tag{3.6}$$

3.2.4 MULTIPLICATION

The product of a scalar c and a quaternion q is defined in a straightforward manner:

$$cq = cq_0 + \boldsymbol{i}\,cq_1 + \boldsymbol{j}\,cq_2 + \boldsymbol{k}cq_3. \tag{3.7}$$

The product of two quaternions $p = p_0 + \boldsymbol{p}$ and $q = q_0 + \boldsymbol{q}$ can be written in a concise form:

$$pq = p_0q_0 - \boldsymbol{p}\cdot\boldsymbol{q} + p_0\boldsymbol{q} + q_0\boldsymbol{p} + \boldsymbol{p}\times\boldsymbol{q}. \tag{3.8}$$

The detailed derivation for Eq. (3.8) is available in [57]. Note that the multiplication of quaternions is not commutative ($pq \neq qp$):

$$qp = q_0p_0 - \boldsymbol{q}\cdot\boldsymbol{p} + q_0\boldsymbol{p} + p_0\boldsymbol{q} + \boldsymbol{q}\times\boldsymbol{p}, \tag{3.9}$$

where the first two components ($p_0q_0 - \boldsymbol{p}\cdot\boldsymbol{q}$) of pq and ($q_0p_0 - \boldsymbol{q}\cdot\boldsymbol{p}$) of qp make up the scalar portion of the product, and the other three terms ($p_0\boldsymbol{q} + q_0\boldsymbol{p} + \boldsymbol{p}\times\boldsymbol{q}$) and ($q_0\boldsymbol{p} + p_0\boldsymbol{q} + \boldsymbol{q}\times\boldsymbol{p}$) comprise the vector portion.

3.2.5 COMPLEX CONJUGATE

Quaternions have similar properties with ordinary complex numbers. The complex conjugate of a quaternion $q = q_0 + \boldsymbol{q}$ is defined as $q^* = q_0 - \boldsymbol{q}$. Given any two quaternions p and q,

$$(pq)^* = q^*p^*. \tag{3.10}$$

The sum of a quaternion and its complex conjugate is a scalar:

$$q + q^* = (q_0 + \boldsymbol{q}) + (q_0 - \boldsymbol{q}) = 2q_0. \tag{3.11}$$

3.2.6 NORM

The norm of a quaternion q is the length of q, and is defined by

$$N(q) = \sqrt{q^*q}. \tag{3.12}$$

From the multiplication definition in Eq. (3.8), substituting $p = q^*$, we can obtain

$$\begin{aligned} N^2(q) &= q_0q_0 - (-\boldsymbol{q})\cdot\boldsymbol{q} + q_0\boldsymbol{q} + (-\boldsymbol{q})q_0 + (-\boldsymbol{q})\times\boldsymbol{q} \\ &= q_0^2 + \boldsymbol{q}\cdot\boldsymbol{q} \\ &= q_0^2 + q_1^2 + q_2^2 + q_3^2 = |q|^2. \end{aligned} \tag{3.13}$$

A unit quaternion is one in which the norm is equal to one.

3.2.7 INVERSE

Every non-zero quaternion has a multiplicative inverse, and it satisfies

$$q^{-1}q = qq^{-1} = 1. \tag{3.14}$$

If we multiply the complex conjugate q^* to the right and the left of Eq. (3.14), we can obtain

$$q^{-1}qq^* = q^*. \tag{3.15}$$

Due to $qq^* = N^2(q)$,

$$q^{-1} = \frac{q^*}{N^2(q)} = \frac{q^*}{|q|^2}. \tag{3.16}$$

If q is a unit quaternion, then the inverse is equivalent with the complex conjugate:

$$q^{-1} = q^*. \tag{3.17}$$

Note that Eq. (3.17) is analogous to the inverse of a rotation matrix $\boldsymbol{M}$, where $\boldsymbol{M}^{-1} = \boldsymbol{M}^T$.

Finally, a quaternion in which the real part is zero is called a *pure* quaternion.

3.3 ORIENTATION REPRESENTATIONS

One of the most building blocks in many computer applications is a mathematical model relating the orientation of a rotating body frame relative to a fixed reference frame. The fixed reference frame is often an earth-based reference frame, and the orientation of one frame of reference to the other can be described as a rotation. There are several ways to describe rotations. Euler angles, rotation matrices, and quaternion representations are described in the following subsections.

3.3.1 EULER ANGLES AND ROTATION MATRICES

Consider the reference frames in Fig. 3.13. The fixed reference frame is the global frame based on the earth reference (denoting $\mathcal{G}$), whereas the rotating reference frame is the sensor reference frame (denoting $\mathcal{S}$) that we wish to estimate the sensor device's orientation compared to the global reference frame. First, let us arbitrarily choose the phone screen XY coordinates as a rotating reference frame for discussion. We can intuitively see that any x, y points (z assumed zero) on the phone screen will map to some x, y, z point in the global reference frame. Position offset $\boldsymbol{r}$ in Fig. 3.13 does not impact our computation of orientation, and we can collapse the frames, as shown in Fig. 3.14. Now let us remove the phone from the figure and focus just on points in the XY plane. Figure 3.15 shows the global frame $\mathcal{G}$ is rotated into the sensor frame $\mathcal{S}$ by an angle ψ. It allows us to map any point $\boldsymbol{v}$ in the global reference frame $\mathcal{G}$ to a point in the sensor reference frame $\mathcal{S}$ with a simple linear transformation. The mapping is quite straightforward requiring only high school trigonometry. $({}^{\mathcal{S}}x, {}^{\mathcal{S}}y)$ can be computed from

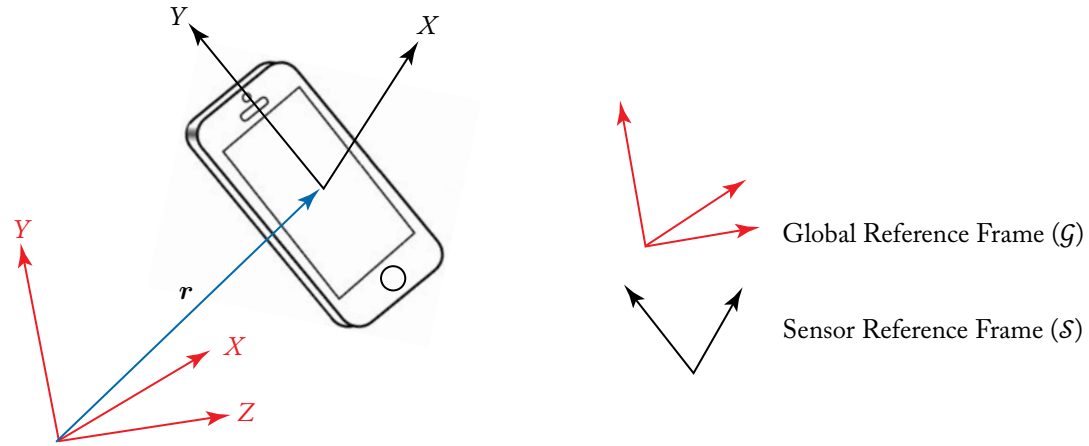

Figure 3.13: Reference frames.

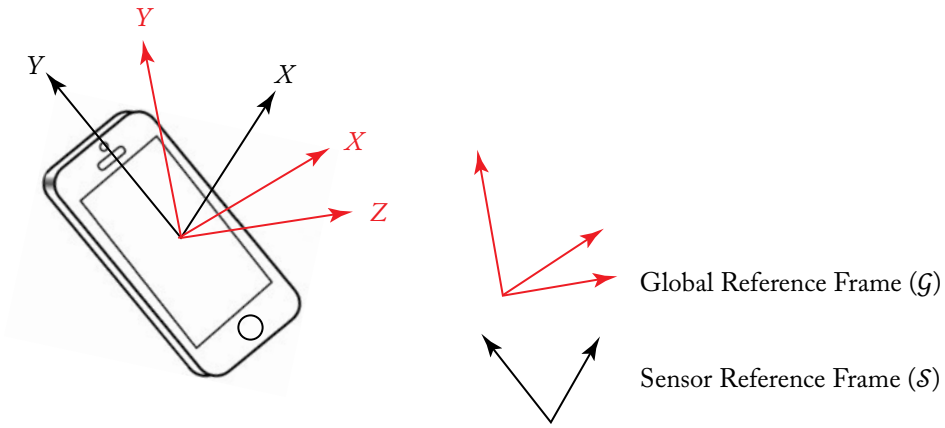

Figure 3.14: Collapsed frames.

$({}^{\mathcal{G}}x, {}^{\mathcal{G}}y)$:

$$\begin{aligned} {}^{\mathcal{S}}x &= {}^{\mathcal{G}}x\cos\psi + {}^{\mathcal{G}}y\sin\psi \\ {}^{\mathcal{S}}y &= -{}^{\mathcal{G}}x\sin\psi + {}^{\mathcal{G}}y\cos\psi. \end{aligned} \tag{3.18}$$

In a matrix form,

$${}^{\mathcal{S}}\boldsymbol{v} = \boldsymbol{M}_{\psi}\,{}^{\mathcal{G}}\boldsymbol{v}, \tag{3.19}$$

where ${}^{\mathcal{G}}\boldsymbol{v}$ and ${}^{\mathcal{S}}\boldsymbol{v}$ are of the vector form $[x\ y]^T$. Since we consider that the axes were rotated by ψ as shown in Fig. 3.15, the corresponding angle of the vector $\boldsymbol{v}$ is reduced by ψ. The rotation

matrix is

$$\boldsymbol{M}_{\psi} = \begin{bmatrix} \cos\psi & \sin\psi \\ -\sin\psi & \cos\psi \end{bmatrix}. \tag{3.20}$$

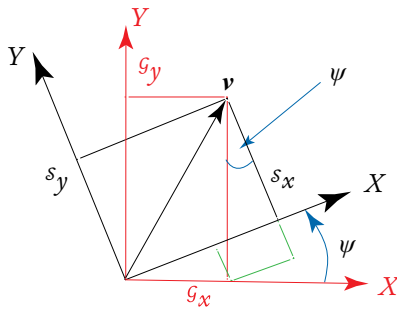

Figure 3.15: 2D space for Fig. 3.14.

The analysis also extends naturally to three dimensions. If ${}^{\mathcal{G}}\boldsymbol{v}$ and ${}^{\mathcal{S}}\boldsymbol{v}$ are of the vector form $[x\ y\ z]^T$, then the rotation matrices will be:

$$\boldsymbol{M}_{\psi} = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad \boldsymbol{M}_{-\psi} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.21}$$

Note that $\boldsymbol{M}_{-\psi}$ is the transpose of $\boldsymbol{M}_{\psi}$ (i.e., $\boldsymbol{M}_{-\psi} = \boldsymbol{M}_{\psi}^T$). Also, it is clear that $\boldsymbol{M}_{\psi}\boldsymbol{M}_{-\psi} = \boldsymbol{I}$. This implies that inverse of a rotation matrix is its own transpose: $\boldsymbol{M}_{\psi}^T = \boldsymbol{M}_{\psi}^{-1}$. Similar rotation matrices hold for XZ and YZ planes. The rotation matrices are:

$$\boldsymbol{M}_{\theta} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}, \qquad \boldsymbol{M}_{-\theta} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{3.22}$$

$$\boldsymbol{M}_{\phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}, \qquad \boldsymbol{M}_{-\phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}. \tag{3.23}$$

The angles ϕ, θ, ψ are called roll, pitch, and yaw, respectively. Within a 3D right-handed Cartesian coordinate system, any rotational mapping in 3D space can be defined as a sequence of rotations defined by these angles. For example, a rotation from earth to body frame about Z,

then Y, and then X axes can be represented by yaw-pitch-roll:

$$\begin{aligned}\boldsymbol{M}_{RPY} &= \boldsymbol{M}_{\phi}\boldsymbol{M}_{\theta}\boldsymbol{M}_{\psi} \\ &= \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\cos\theta \end{bmatrix}.\end{aligned} \tag{3.24}$$

Note that the composite rotation matrix is computed simply by multiplying the three individual matrices in the specified order. Consistent with the discussion above, the inverse of this expression is simply its transpose:

$$\boldsymbol{M}_{YPR} = \boldsymbol{M}_{-\psi}\boldsymbol{M}_{-\theta}\boldsymbol{M}_{-\phi} = \boldsymbol{M}_{RPY}^{T}. \tag{3.25}$$

Collectively, ψ, θ, ϕ are known as *Euler angles*. The subscript "RPY" refers to roll-pitch-yaw. Euler angles are sometimes sub-divided into *Tait–Bryan* angles, in which rotations occur about all three axes, and *proper Euler angles*, in which the first and third axes of rotation are the same. Regardless of the type, it is important to specify the order of Euler rotations which is significant. The right-most rotation is performed first, consistent with the matrix operations that will be required to implement the rotation. Possible variants are described in Table 3.2.

Table 3.2: Euler angles

Alpha	Angles	Comments	
YRP	$\psi - \phi - \theta$	Tait-Bryan angles	All of these are sometimes referred to simply as "Euler Angles"
YPR	$\psi - \theta - \phi$		
PYR	$\theta - \psi - \phi$		
PRY	$\theta - \phi - \psi$		
RYP	$\phi - \psi - \theta$		
RPY	$\phi - \theta - \psi$		
RYR	$\phi - \psi - \phi$	Proper Euler Angles	
RPR	$\phi - \theta - \phi$		
PYP	$\theta - \psi - \theta$		
PRP	$\theta - \phi - \theta$		
YRY	$\psi - \phi - \psi$		
YPY	$\psi - \theta - \psi$		

Table 3.2 clearly shows that Euler-based orientation definitions are not unique. Any given orientation can be described by numerous, varying, Euler combinations. However, the resulting

3-dimensional rotation matrices ARE unique. Equivalent Euler-based orientations will result in the same numerical rotation matrix.

There are some key points we need to take away, before we move onto the quaternion based method.

- The order of rotation matters.
- There are almost as many notations for Euler angles as the references in which they appear. The notation should be specific.
- There may be multiple Euler angle combinations which map to the same physical rotation. They are not unique.
- There are typically limits on the range of Euler rotation for each axis ($\pm\pi$ or $\pm\pi/2$). These effect how the total rotation is spread across the three angles. They also tend to introduce discontinuities when Euler angles are computed over time.
- Each Euler triad will map to 3×3 rotation matrix which is unique.
- Each rotation matrix requires nine numbers in the form which may result in significant consumption of memory and computation cost.

3.3.2 QUATERNIONS

Quaternion as Rotation

As discussed, if there is a reference orientation defined for any object, then we can define its current orientation as some rotation relative to that reference. Tracking orientation over time is then equivalent to tracking rotation from that reference over time. Because Euler angles are relatively easy to visualize, they have been widely used in many fields. But because of the shortcomings listed in Section 3.3.1, we introduce another representation: the *unit quaternion*. We already reviewed the basic algebra of quaternions in Section 3.2. Here, we apply it for rotational operation in 3D space.

Consider the rotation of a simple rigid body as shown in Fig. 3.16 (shown here as a cylinder centered about an axis of rotation). The cylinder is rotated such that a point on its surface originally at $\boldsymbol{v}$ is rotated to point $\boldsymbol{w}$ in space. It is easy to see that the movement of the cylinder is a rotation equal to angle 2α about the axis of rotation $\boldsymbol{u}$. Let us consider a normalized quaternion

$$q = q_0 + \boldsymbol{q}, \tag{3.26}$$

where $q_0^2 + |\boldsymbol{q}|^2 = 1$. Using the Pythagorean identity as a guide, for a rotation angle 2α, we define

$$q_0^2 = \cos^2\alpha, \qquad |\boldsymbol{q}|^2 = \sin^2\alpha, \tag{3.27}$$

where the angle α is uniquely defined in $-\pi/2 < \alpha \leq \pi/2$. Then, the quaternion q can be defined in terms of the angle α and the normalized rotation axis $\boldsymbol{u}$:

$$q = q_0 + \boldsymbol{q} = \cos\alpha + \boldsymbol{u}\sin\alpha, \tag{3.28}$$

where $\boldsymbol{u} = \frac{\boldsymbol{q}}{|\boldsymbol{q}|} = \frac{\boldsymbol{q}}{\sin\alpha}$. Equation 3.28 allows us to express any rotation of angle 2α about rotation axis $\boldsymbol{u}$ in quaternion form.

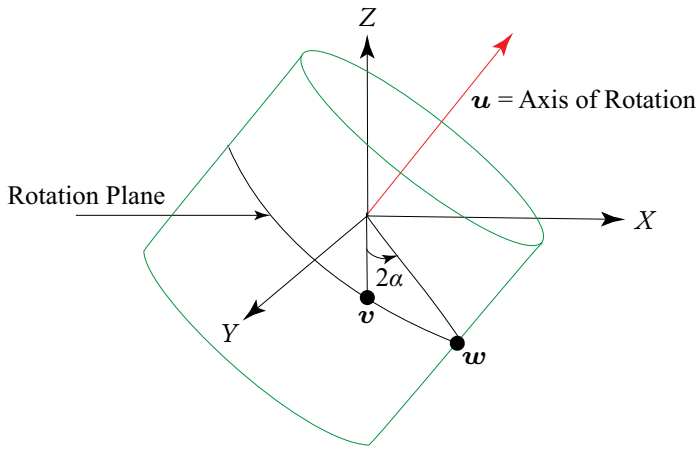

Figure 3.16: Overlay of Cartesian coordinates onto a rotational cylinder.

We can also use quaternions to create a rotation operator capable of rotating one vector into another. If we treat $\boldsymbol{v}$ and $\boldsymbol{w}$ as pure quaternions with zero real components, then we can rotate $\boldsymbol{v}$ into $\boldsymbol{w}$ using the quaternion rotation operator:

$$\boldsymbol{w} = L_q(\boldsymbol{v}) = q\boldsymbol{v}q^*. \tag{3.29}$$

Alternately,

$$L_{q^*}(\boldsymbol{v}) = q^*\boldsymbol{v}q \tag{3.30}$$

can be viewed either as a rotation in the opposite direction, or a rotation of the reference frame with respect to the vector $\boldsymbol{v}$.[2]

Refer to [57] for additional details regarding Eqs. (3.29) and (3.30).

Vector Proof

Figure 3.17 provides the basis for a more detailed discussion of the quaternion rotation operator. Let $\boldsymbol{v}$ and $\boldsymbol{w}$ be the vectors for the initial and the rotated locations, respectively, in 3D space. Let

[2] [56] refers to $L_q(\boldsymbol{v})$ as a "point rotation" and $L_{q^*}(\boldsymbol{v})$ as a "frame rotation."

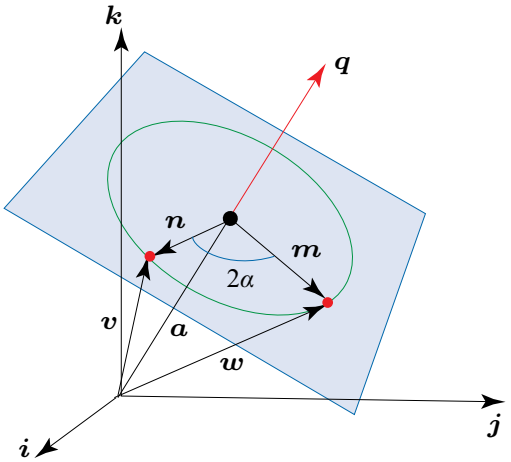

Figure 3.17: Geometric interpretation for quaternion rotation operation.

$\boldsymbol{n}$ and $\boldsymbol{m}$ be the vectors located on the rotation plane, as shown in Fig. 3.17. Then the original vector $\boldsymbol{v}$ follows:

$$\boldsymbol{v} = \boldsymbol{a} + \boldsymbol{n}. \tag{3.31}$$

The quaternion rotation operator becomes

$$\boldsymbol{w} = q\boldsymbol{v}q^* = q\boldsymbol{a}q^* + q\boldsymbol{n}q^* = \boldsymbol{a} + q\boldsymbol{n}q^* \tag{3.32}$$

where the third equality is because any vector on the axis of rotation $\boldsymbol{q}$ should be invariant. The second term $q\boldsymbol{n}q^*$ of Eq. (3.32) can be given by

$$\begin{aligned} q\boldsymbol{n}q^* &= (q_0 + \boldsymbol{q})\,(0 + \boldsymbol{n})\,(q_0 - \boldsymbol{q}) \\ &= \left(q_0^2 - |\boldsymbol{q}|^2\right)\boldsymbol{n} + 2\,(\boldsymbol{q}\cdot\boldsymbol{n})\,\boldsymbol{q} + 2q_0(\boldsymbol{q}\times\boldsymbol{n}) \\ &= \left(q_0^2 - |\boldsymbol{q}|^2\right)\boldsymbol{n} + 2q_0|\boldsymbol{q}|(\boldsymbol{u}\times\boldsymbol{n}), \end{aligned} \tag{3.33}$$

where the second equality is based on the multiplication property in Section 3.2.4 and the property of cross product $\boldsymbol{a}\times(\boldsymbol{b}\times\boldsymbol{c}) = \boldsymbol{b}(\boldsymbol{a}\cdot\boldsymbol{c}) - \boldsymbol{c}(\boldsymbol{a}\cdot\boldsymbol{b})$. The third term is due to $\boldsymbol{q}\cdot\boldsymbol{n} = 0$ and $\boldsymbol{u} = \boldsymbol{q}/|\boldsymbol{q}|$. By the definitions of q_0^2 and $|\boldsymbol{q}|^2$ in Eq. (3.27), Eq. (3.33) can be rewritten in the trigonometric form as

$$\begin{aligned} \left(q_0^2 - |\boldsymbol{q}|^2\right)\boldsymbol{n} + 2q_0|\boldsymbol{q}|(\boldsymbol{u}\times\boldsymbol{n}) &= \left(\cos^2\alpha - \sin^2\alpha\right)\boldsymbol{n} + (2\cos\alpha\sin\alpha)\,(\boldsymbol{u}\times\boldsymbol{n}) \\ &= (\cos 2\alpha)\boldsymbol{n} + (\sin 2\alpha)(\boldsymbol{u}\times\boldsymbol{n}) \\ &= \boldsymbol{m}. \end{aligned} \tag{3.34}$$

From (3.33) and (3.34) we have $q\boldsymbol{n}q^* = \boldsymbol{m}$. Substituting this into (3.32), we obtain

$$\boldsymbol{w} = \boldsymbol{a} + \boldsymbol{m}. \tag{3.35}$$

This is also observed in Fig. 3.17. The resulting rotation angle is 2α if the quaternion is defined as $q = \cos\alpha + \boldsymbol{u}\sin\alpha$.

Sequences of Rotations

One of the great benefits of representing orientations and rotations as quaternions is that a sequence of rotations can be modeled using quaternion multiplication. Let us assume starting vector $\boldsymbol{u}$ is rotated using quaternion p, and then the result rotated using quaternion q. The sequence of events could be viewed as shown in Fig. 3.18.

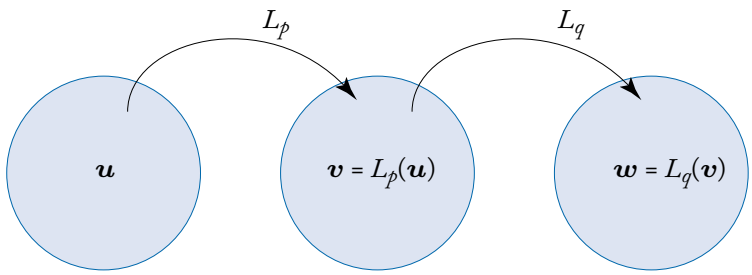

Figure 3.18: Modeling sequential rotations.

$$\boldsymbol{v} = L_p(\boldsymbol{u}) = p\boldsymbol{u}p^* \tag{3.36}$$
$$\boldsymbol{w} = L_q(\boldsymbol{v}) = q\boldsymbol{v}q^* = q(p\boldsymbol{u}p^*)q^* \tag{3.37}$$
$$\boldsymbol{w} = (qp)\boldsymbol{u}(p^*q^*) = (qp)\boldsymbol{u}(qp)^* \tag{3.38}$$
$$\boldsymbol{w} = L_{qp}(\boldsymbol{u}) \tag{3.39}$$

Equation (3.39) shows that we can simply pre-multiply rotation quaternions before applying the composite rotation operator to obtain the final result. This results in dramatic efficiencies compared to other computation methods.

3.3.3 CONVERSIONS BETWEEN REPRESENTATIONS

It is easy to convert between the various orientation representations.

Rotation Matrices to Euler Angles

Equations (3.24) and (3.25) have already shown how to move from Euler angles to rotation matrices. Now we describe the inverse: from rotation matrices to Euler angles. Let $\boldsymbol{M}$ be a

rotation matrix that is given by

$$\boldsymbol{M} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}. \tag{3.40}$$

The roll-pitch-yaw rotation matrix is

$$\begin{aligned} \boldsymbol{M}_{RPY} &= \boldsymbol{M}_{\phi}\boldsymbol{M}_{\theta}\boldsymbol{M}_{\psi} \\ &= \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\cos\theta \end{bmatrix} \end{aligned} \tag{3.24}$$

we can write the following by inspection:

$$\theta = -\sin^{-1}(m_{12}) \tag{3.41}$$

$$\phi = \cos^{-1}(m_{33}/\cos\theta) \tag{3.42}$$

$$\psi = \cos^{-1}(m_{11}/\cos\theta) \tag{3.43}$$

subject to m_{12} not equal to 1. One only needs to use m_{21} in place of m_{12} in the expression for θ to extract Euler angles from $\boldsymbol{M}_{YPR}$ (recall $\boldsymbol{M}_{YPR} = \boldsymbol{M}_{RPY}^{T}$).

Quaternions to Rotation Matrices

Consider the quaternion rotations of the form:

$$\boldsymbol{w} = L_q(\boldsymbol{v}) = q\boldsymbol{v}q^* \tag{3.44}$$

which can be converted to rotation matrix

$$\boldsymbol{M}_{q\boldsymbol{v}q^*} = \begin{bmatrix} 2q_0^2 - 1 + 2q_1^2 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3 + 2q_0q_2 \\ 2q_1q_2 + 2q_0q_3 & 2q_0^2 - 1 + 2q_2^2 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_0q_2 & 2q_2q_3 + 2q_0q_1 & 2q_0^2 - 1 + 2q_3^2 \end{bmatrix}. \tag{3.45}$$

An equivalent form of (3.45) is:

$$\boldsymbol{M}_{q\boldsymbol{v}q^*} = \begin{bmatrix} 1 - 2q_2^2 - 2q_3^2 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3 + 2q_0q_2 \\ 2q_1q_2 + 2q_0q_3 & 1 - 2q_1^2 - 2q_3^2 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_0q_2 & 2q_2q_3 + 2q_0q_1 & 1 - 2q_1^2 - 2q_2^2 \end{bmatrix}. \tag{3.46}$$

Alternatively, we can also consider the other quaternion rotations of the form:

$$L_{q^*}(\boldsymbol{v}) = q^*\boldsymbol{v}q \tag{3.30}$$

which can be converted to rotation matrix

$$\boldsymbol{M}_{q^* \boldsymbol{v} q} = \begin{bmatrix} 2q_0^2 - 1 + 2q_1^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & 2q_0^2 - 1 + 2q_2^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & 2q_0^2 - 1 + 2q_3^2 \end{bmatrix}. \tag{3.47}$$

An equivalent form of (3.47) is:

$$\boldsymbol{M}_{q^* \boldsymbol{v} q} = \begin{bmatrix} 1 - 2q_2^2 - 2q_3^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & 1 - 2q_1^2 - 2q_3^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & 1 - 2q_1^2 - 2q_2^2 \end{bmatrix}. \tag{3.48}$$

As expected, (3.45) and (3.46) are simply the transpose of (3.47) and (3.48), respectively: $\boldsymbol{M} = \boldsymbol{Q}^T$.

Rotation Matrices to Quaternions

Rotation matrices can be converted to quaternion for rotations of the form:

$$L_{q^*}(\boldsymbol{v}) = q^* \boldsymbol{v} q \tag{3.30}$$

using (3.40) and (3.47)

$$\boldsymbol{M} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} = \begin{bmatrix} 2q_0^2 - 1 + 2q_1^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & 2q_0^2 - 1 + 2q_2^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & 2q_0^2 - 1 + 2q_3^2 \end{bmatrix} \tag{3.49}$$

$$q_0 = \frac{1}{2}\sqrt{m_{11} + m_{22} + m_{33} + 1} \tag{3.50}$$

$$q_1 = (m_{23} - m_{32}) / (4q_0) \tag{3.51}$$

$$q_2 = (m_{31} - m_{13}) / (4q_0) \tag{3.52}$$

$$q_3 = (m_{12} - m_{21}) / (4q_0). \tag{3.53}$$

Simply invert q_1, q_2, and q_3 to support rotations of the form: $\boldsymbol{w} = L_q(\boldsymbol{v}) = q \boldsymbol{v} q^*$.

3.3.4 ORIENTATION REPRESENTATION COMPARISON

Table 3.3 compares the advantages and disadvantages when quaternions or rotation matrices are used.

Table 3.3: Quaternion vs. rotation matrix

Topic	Quaternion	Rotation Matrix
Storage	Requires 16 bytes of storage in single-precision floating point (4 elements at 4 bytes each).	Requires 36 bytes of storage (9 elements at 4 bytes each).
Computation (2 sequential rotations)	4 elements each requiring 4 multiplies and 3 additions = 28 operations.	9 elements, each requiring 3 multiplies and 2 additions = 45 operations.
Vector Rotation	Rotating a vector by pre- and post-multiplication of quaternion requires 56 operations.	Rotating a vector via rotation matrix requires 15 operations (3 elements each requiring 3 multiplies and 2 additions).
Discontinuities	Generally, we force the scalar part of the quaternion to be positive, which can cause a discontinuity in the rotation axis (it flips).	None.
Conversion	From rotation matrix $\boldsymbol{M} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}$ we have $q_0 = 0.5\sqrt{m_{11} + m_{22}\ m_{33} + 1}$ $q_1 = \frac{m_{23} - m_{32}}{4q_0}, q_2 = \frac{m_{31} - m_{13}}{4q_0},$ $q_3 = \frac{m_{12} - m_{21}}{4q_0}.$	Rotation matrix: $\boldsymbol{M} = \begin{bmatrix} 1 - 2q_2^2 - 2q_3^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & 1 - 2q_1^2 - 2q_3^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & 1 - 2q_1^2 - 2q_2^2 \end{bmatrix}$

Computing the Quaternion for the Angle Between Two Vectors

Given two vectors $\boldsymbol{v}$ and $\boldsymbol{w}$, we can obtain the rotation quaternion q to rotate $\boldsymbol{v}$ into $\boldsymbol{w}$. The rotation angle 2α between $\boldsymbol{v}$ and $\boldsymbol{w}$ can be determined by

$$\cos 2\alpha = 2\cos^2\alpha - 1 = \frac{\boldsymbol{v} \cdot \boldsymbol{w}}{|\boldsymbol{v}||\boldsymbol{w}|}. \tag{3.54}$$

The scalar component q_0 of q then be obtained by

$$q_0 = \cos\alpha = \sqrt{\frac{|\boldsymbol{v}||\boldsymbol{w}| + \boldsymbol{v}\cdot\boldsymbol{w}}{2|\boldsymbol{v}||\boldsymbol{w}|}}. \tag{3.55}$$

The vector component $\boldsymbol{q}$ of q with the axis of rotation $\boldsymbol{u}$ is

$$\boldsymbol{u}\sin 2\alpha = 2\boldsymbol{u}\sin\alpha\cos\alpha = \frac{-\boldsymbol{v}\times\boldsymbol{w}}{|\boldsymbol{v}||\boldsymbol{w}|}, \tag{3.56}$$

where the negative sign in Eq. (3.56) is because the rotation axis for a coordinate system is calculated rather than the axis required to rotate the vector $\boldsymbol{v}$ onto $\boldsymbol{w}$ in a fixed coordinate system. Note that the angle 2α to rotate the vector $\boldsymbol{v}$ onto $\boldsymbol{w}$ is negated when a coordinate system is rotated. By substituting Eq. (3.55) into Eq. (3.56), we obtain the vector component $\boldsymbol{q}$ which is given by

$$\boldsymbol{q} = [q_1\, q_2\, q_3]^T = \boldsymbol{u}\sin\alpha = \frac{-\boldsymbol{v}\times\boldsymbol{w}}{2|\boldsymbol{v}||\boldsymbol{w}|\sqrt{\frac{|\boldsymbol{v}||\boldsymbol{w}|+\boldsymbol{v}\cdot\boldsymbol{w}}{2|\boldsymbol{v}||\boldsymbol{w}|}}}. \tag{3.57}$$

Equations (3.55) and (3.57) result in the required rotation quaternion:

$$\begin{aligned} q &= \cos\alpha + \boldsymbol{u}\sin\alpha \\ &= \sqrt{\frac{|\boldsymbol{v}||\boldsymbol{w}| + \boldsymbol{v}\cdot\boldsymbol{w}}{2|\boldsymbol{v}||\boldsymbol{w}|}} - \frac{\boldsymbol{v}\times\boldsymbol{w}}{2|\boldsymbol{v}||\boldsymbol{w}|\sqrt{\frac{|\boldsymbol{v}||\boldsymbol{w}|+\boldsymbol{v}\cdot\boldsymbol{w}}{2|\boldsymbol{v}||\boldsymbol{w}|}}} = \frac{|\boldsymbol{v}||\boldsymbol{w}| + \boldsymbol{v}\cdot\boldsymbol{w} - \boldsymbol{v}\times\boldsymbol{w}}{\sqrt{2|\boldsymbol{v}||\boldsymbol{w}|(|\boldsymbol{v}||\boldsymbol{w}| + \boldsymbol{v}\cdot\boldsymbol{w})}}. \end{aligned} \tag{3.58}$$

In the specific case of $2\alpha = 180°$, however, the quaternion in Eq. (3.58) is undefined because $|\boldsymbol{v}||\boldsymbol{w}| + \boldsymbol{v}\cdot\boldsymbol{w} = 0$. One solution for this problem is, according to [57],

$$\boldsymbol{q} = \frac{1}{\sqrt{(v_y - v_z)^2 + (v_z - v_x)^2 + (v_x - v_y)^2}} \begin{bmatrix} v_y - v_z \\ v_z - v_x \\ v_x - v_y) \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}. \tag{3.59}$$

If the two vectors $\boldsymbol{v}$ and $\boldsymbol{w}$ are assumed unit norm, Eq. (3.58) is simplified as

$$q = \frac{1 + \boldsymbol{v}\cdot\boldsymbol{w} - \boldsymbol{v}\times\boldsymbol{w}}{\sqrt{2(1 + \boldsymbol{v}\cdot\boldsymbol{w})}} = \frac{1}{\sqrt{2}}\sqrt{1 + \boldsymbol{v}\cdot\boldsymbol{w}} - \frac{\boldsymbol{v}\times\boldsymbol{w}}{\sqrt{1 + \boldsymbol{v}\cdot\boldsymbol{w}}} \quad \text{for} \quad |\boldsymbol{v}| = |\boldsymbol{w}| = 1. \tag{3.60}$$

3.4 VIRTUAL GYROSCOPE

We have explored some mathematical models to represent the orientation of a sensor body. As a sensor fusion application, we will build a virtual gyroscope using data from a 3-axis accelerometer and 3-axis magnetometer. Reasons we might build a virtual gyroscope instead of an actual one include *cost* and *cost*. The first cost is financial. Gyro sensors tend to be more expensive

than the other two sensors. Eliminating them is attractive for that reason. The second cost is power. The power consumed by a typical accelerometer and magnetometer pair is significantly less than that consumed by a MEMS gyroscope. Nonetheless, the downside of a virtual gyro is that it is sensitive to linear acceleration and uncorrected magnetic interference. If either of those is present, a physical gyro sensor will be wanted. Angular rate can be defined as change in orientation per unit time. We will take the derivative of one of the orientation representations. Recall the following notations:

$^{\mathcal{G}}\boldsymbol{v}$: vector $\boldsymbol{v}$ measured in a (fixed) global reference frame.
$^{\mathcal{S}}\boldsymbol{v}$: vector $\boldsymbol{v}$ measured in a (rotating) sensor reference frame.
$\boldsymbol{M}_t$: rotation matrix which takes $^{\mathcal{G}}\boldsymbol{v}$ into $^{\mathcal{S}}\boldsymbol{v}$ at time t.
$\boldsymbol{\omega}$: angular rate through the rotation.

Then at any time t,

$$^{\mathcal{S}}\boldsymbol{v} = \boldsymbol{M}_t\,^{\mathcal{G}}\boldsymbol{v}. \tag{3.61}$$

Differentiating both sides becomes

$$\frac{^{\mathcal{S}}d\boldsymbol{v}}{dt} = \frac{d\boldsymbol{M}_t}{dt}\,^{\mathcal{G}}\boldsymbol{v} + \boldsymbol{M}_t\frac{^{\mathcal{G}}d\boldsymbol{v}}{dt}. \tag{3.62}$$

It is assumed that there exists an inverse matrix of $\boldsymbol{M}_t$. Under the restrictions on no linear acceleration or magnetic interference (i.e., $^{\mathcal{G}}d\boldsymbol{v}/dt = 0$), substituting Eq. (3.61) into Eq. (3.62) yields

$$\frac{^{\mathcal{S}}d\boldsymbol{v}}{dt} = \frac{d\boldsymbol{M}_t}{dt}\boldsymbol{M}_t^{-1}\,^{\mathcal{S}}\boldsymbol{v}. \tag{3.63}$$

In Section 2.1.1, we learned about the transport theorem, which describes the rate of change of a vector in a rotating reference frame:

$$\frac{^{\mathcal{G}}d\boldsymbol{v}}{dt} = \frac{^{\mathcal{S}}d\boldsymbol{v}}{dt} - \boldsymbol{\omega} \times {}^{\mathcal{S}}\boldsymbol{v}. \tag{3.64}$$

We've inverted the omega term from Eq. (2.12) to account for a change in perspective. We want $\boldsymbol{\omega}$ from the sensor's perspective. Given $^{\mathcal{G}}d\boldsymbol{v}/dt = 0$, we obtain

$$\frac{^{\mathcal{S}}d\boldsymbol{v}}{dt} = \boldsymbol{\omega} \times {}^{\mathcal{S}}\boldsymbol{v}. \tag{3.65}$$

Equating Eqs. (3.63) and (3.65), we have

$$\boldsymbol{\omega} \times {}^{\mathcal{S}}\boldsymbol{v} = \frac{d\boldsymbol{M}_t}{dt}\boldsymbol{M}_t^{-1}\,^{\mathcal{S}}\boldsymbol{v}, \tag{3.66}$$

where the cross product in the LHS can be defined as the product of a skew-symmetric matrix and a vector [143, 163]:[3]

$$\begin{aligned} \boldsymbol{\omega} \times {}^{\mathcal{S}}\boldsymbol{v} &= [\boldsymbol{\omega}]_{\times}{}^{\mathcal{S}}\boldsymbol{v} \\ &= \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} {}^{\mathcal{S}}\boldsymbol{v}. \end{aligned} \tag{3.67}$$

Assume the change of angle is relatively small so that we can approximate the derivative of the RHS in Eq. (3.66):

$$\frac{d\,\boldsymbol{M}_t}{dt} \approx \frac{1}{\Delta t}\left(\boldsymbol{M}_{t+1} - \boldsymbol{M}_t\right), \tag{3.68}$$

where Δt is the time interval between orientation samples. Recall that for rotation matrices, the transpose is the same as the inverse: $\boldsymbol{M}_t^{-1} = \boldsymbol{M}_t^T$. Therefore, from Eqs. (3.66), (3.67), and (3.68) we obtain

$$[\boldsymbol{\omega}]_{\times} \approx \frac{1}{\Delta t}\left(\boldsymbol{M}_{t+1}\boldsymbol{M}_t^T - \boldsymbol{I}_{3\times 3}\right). \tag{3.69}$$

Define

$$\boldsymbol{\Omega} \triangleq \boldsymbol{M}_{t+1}\boldsymbol{M}_t^T - \boldsymbol{I}_{3\times 3} \approx \begin{bmatrix} 0 & \Omega_{1,2} & \Omega_{1,3} \\ \Omega_{2,1} & 0 & \Omega_{2,3} \\ \Omega_{3,1} & \Omega_{3,2} & 0 \end{bmatrix}, \tag{3.70}$$

where the zero value diagonal elements in $\boldsymbol{\Omega}$ result from small angle approximations since the diagonal terms on $\boldsymbol{M}_{t+1}\boldsymbol{M}_t^T$ will be close to one, which will be canceled by the subtraction of the identity matrix. Then, we can obtain

$$[\boldsymbol{\omega}]_{\times} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \approx \frac{1}{\Delta t}\begin{bmatrix} 0 & \Omega_{1,2} & \Omega_{1,3} \\ \Omega_{2,1} & 0 & \Omega_{2,3} \\ \Omega_{3,1} & \Omega_{3,2} & 0 \end{bmatrix}. \tag{3.71}$$

The angular rate $\boldsymbol{\omega} = [\omega_x\ \omega_y\ \omega_z]^T$ can be obtained by

$$\begin{aligned} \omega_x &\approx \frac{1}{2\Delta t}\left(\Omega_{3,2} - \Omega_{2,3}\right), \\ \omega_y &\approx \frac{1}{2\Delta t}\left(\Omega_{1,3} - \Omega_{3,1}\right), \\ \omega_z &\approx \frac{1}{2\Delta t}\left(\Omega_{2,1} - \Omega_{1,2}\right). \end{aligned} \tag{3.72}$$

[3] https://en.wikipedia.org/wiki/Skew-symmetric_matrix

A similar result can be computed by taking the small-scale approximation of

$$\boldsymbol{M}_{RPY}^{T} = \begin{bmatrix} \cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi \\ \cos\theta\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix}. \quad (3.73)$$

For small ϕ, θ, and ψ, this reduces to:

$$\begin{bmatrix} 1 & -\psi & \theta \\ \psi & 1 & -\phi \\ -\theta & \phi & 1 \end{bmatrix}. \quad (3.74)$$

Take the time derivative and the right-hand side of (3.74) becomes:

$$\begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \quad (3.75)$$

which leads to expressions for $\boldsymbol{\omega}$ consistent with (3.71). This same result should apply regardless ofthe specific Euler sequence used to derive the expression for the starting rotation matrix.

Figure 3.19 shows a MATLAB™ simulation to compare the outputs of a gyroscope and a *virtual gyroscope* based upon accelerometer and magnetometer readings. The results were obtained after adjusting for gyroscope offset and scale factors.

The methods explained above started with the assumption that we already know how to calculate orientation given accelerometer and magnetometer readings. There are many ways to do this.

- Compute roll, pitch, and yaw as described in [58]. Use those values to compute rotation matrices as described in Table 3.3.
- Use the Android getRotationMatrix[4] to compute rotation matrices directly. This method uses a sequence of cross products to arrive at the current orientation.
- Use a solution to Wahba's problem [59] to compute the optimal rotation for each time point.

Whichever technique is used, we need to pay attention to a few details.

- Remember that nonzero linear acceleration and/or uncorrected magnetic interference violate the physical assumptions behind the theory.

[4] http://developer.android.com/reference/android/hardware/SensorManager.html

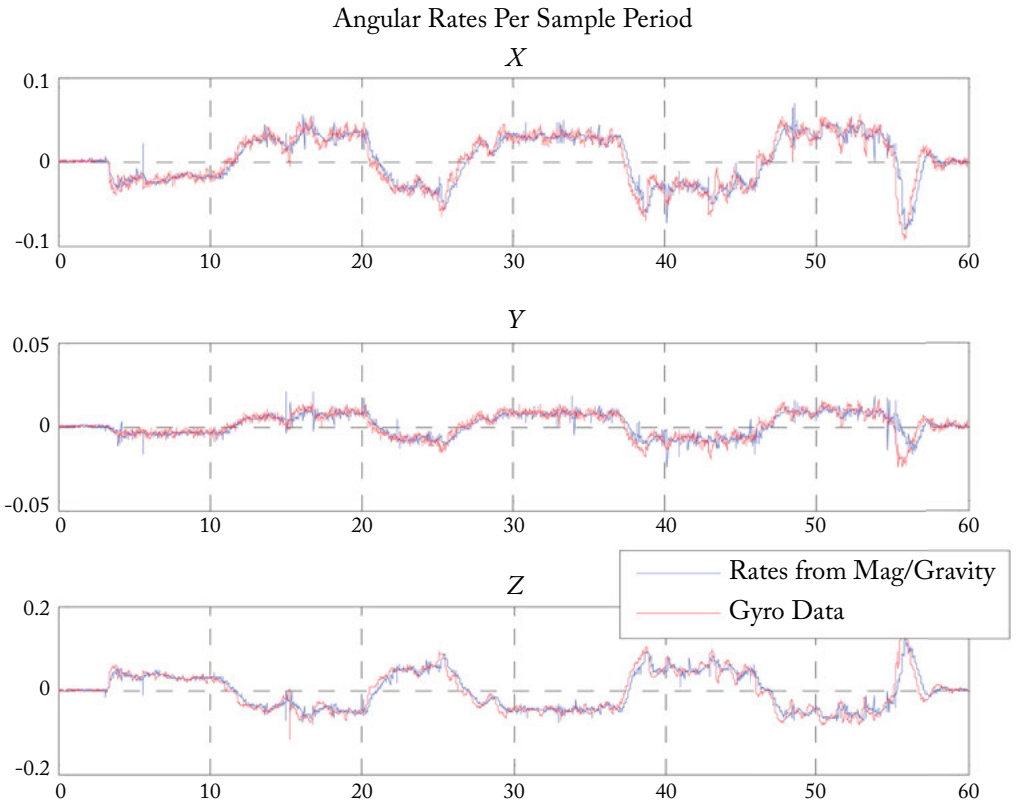

Figure 3.19: Simulation results for virtual gyroscope.

- The expressions shown generally rely on a small angle assumption. That is, the change in orientation from one time step to the next is relatively small. You can encourage this by using a short sampling interval. An alternative method is to discard the slow angle assumption and work with large angles directly.
- Noise in the accelerometer and magnetometer outputs will result in very visible noise in the virtual gyroscope output. Low-pass filtering is preferred.

The virtual gyroscope is an example of sensor fusion applications. There is a certain beauty in the way that nature provides different perspectives of angular motion.

3.5 KALMAN FILTERING FOR ORIENTATION ESTIMATION

3.5.1 INTRODUCTION TO KALMAN FILTERS

We think of a Kalman filter as a least-squares fit to a state-based model. Kalman models [157, 164] predict filter outputs forward in time, and then use actual measurements to correct those estimates.

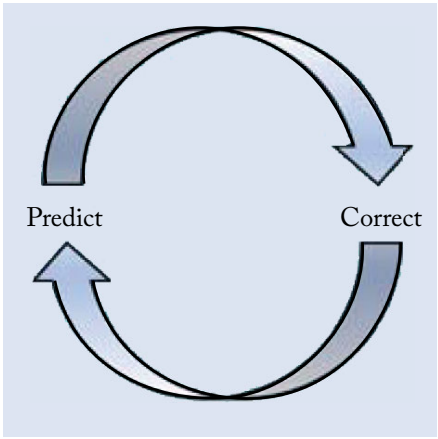

Figure 3.20: Kalman filters predict and correct states over time.

All Kalman filters share two main components. The first is a system model of the form:

$$\boldsymbol{x}_k = \boldsymbol{A}_k \boldsymbol{x}_{k-1} + \boldsymbol{w}_k, \tag{3.76}$$

where $\boldsymbol{x}_k$ is the system state variable at time interval k, $\boldsymbol{A}_k$ is the state transition matrix from one time interval to the next, and $\boldsymbol{w}_k$ is process noise. The second is an observation model of the form:

$$\boldsymbol{z}_k = \boldsymbol{C}_k \boldsymbol{x}_{k-1} + \boldsymbol{v}_k, \tag{3.77}$$

where $\boldsymbol{z}_k$ is an indirect view of the system state at time interval k, $\boldsymbol{C}_k$ describes the relationship between the actual system state (which may not be visible) and observed values. $\boldsymbol{v}_k$ is observation noise.

Kalman filters share a number of characteristics:

- they are a subset of a class of algorithms for optimal state estimation;
- every Kalman filter includes a process model of expected system operation in terms of states;
- every Kalman filter includes an observer model;

- input and output noise must be Gaussian in nature;
- experimental results are "fitted" to the process/observer models; and
- results can be *proved* to be optimum in a least-squares sense.

There are numerous variants of Kalman filter. These include:

- "standard" Kalman filters are used for linear system models;
- "extended" Kalman filters (AKA EKF) linearize non-linear systems;
- "indirect" Kalman filters estimates the state error, not the state itself;
- "complementary" Kalman filters compares multiple input sources; and
- "unscented" Kalman filters (AKA UKF) reduce linearization errors of EKF by focusing on a reduced set of "sigma points" about the mean.

Figure 3.21 and Table 3.4 provide a "cheat sheet" for the basic, linear, Kalman filter. [23] provides additional detail and extends the discussion to some of the filter types in the list above.

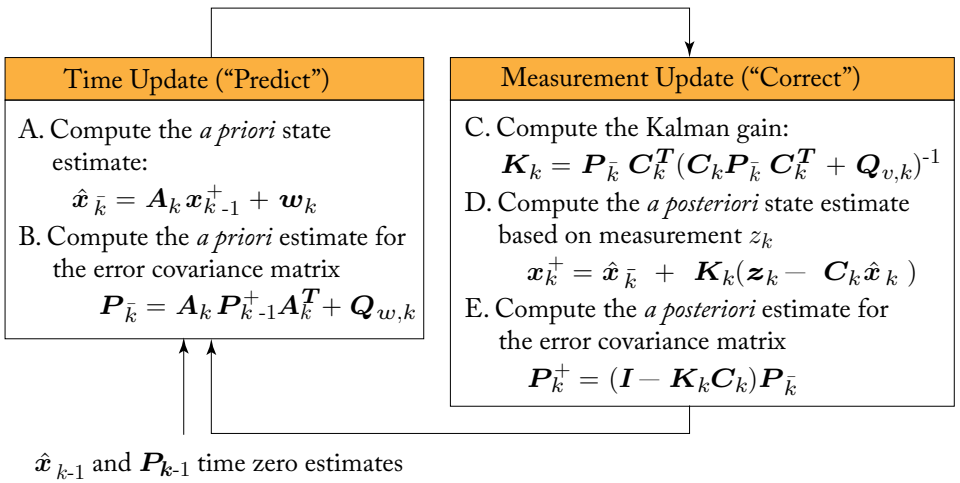

Figure 3.21: Kalman filter processing.[5]

[5] Figure adapted from *An Introduction to the Kalman Filter* by Greg Welch and Gary Bishop. Notation matches NXP application note AN5018.

Table 3.4: Kalman filter variable descriptions

Symbol	Description
$\boldsymbol{A}_k$	Kalman state transition matrix of the process from the state at time interval k to the state at interval k+1.
$\boldsymbol{C}_k$	The measurement matrix relating $\boldsymbol{z}_k$ to $\boldsymbol{x}_k$ at sample k. $\boldsymbol{z}_k = \boldsymbol{C}_k\,\boldsymbol{x}_k + \boldsymbol{v}_k$. In some papers this is denoted $\boldsymbol{H}_k$.
E[ ]	Expectation operator
$\boldsymbol{I}$	Identity matrix
$\boldsymbol{K}_k$	Kalman filter gain matrix at sample k. $\boldsymbol{K}_k = \boldsymbol{P}_k^- \boldsymbol{C}_k^T(\boldsymbol{C}_k \boldsymbol{P}_k^- \boldsymbol{C}_k^T + \boldsymbol{Q}_{v,k})^{-1}$.
$\boldsymbol{P}_k^-$	*A priori* error covariance for interval k. $\boldsymbol{P}_k^- = \boldsymbol{A}_k \boldsymbol{P}_{k-1}^+ \boldsymbol{A}_k^T + \boldsymbol{Q}_{w,k}$. Conceptually, $\boldsymbol{P}_k^- = \mathrm{E}[\hat{\boldsymbol{x}}_{\bar{\varepsilon},k} \hat{\boldsymbol{x}}_{\bar{\varepsilon},k}^T]$.
$\boldsymbol{P}_k^+$	*A posteriori* error covariance for interval k. $\boldsymbol{P}_k^+ = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{C}_k)\boldsymbol{P}_k^-$. Conceptually, $\boldsymbol{P}_k^+ = \mathrm{E}[\hat{\boldsymbol{x}}_{\varepsilon,k}^+ \hat{\boldsymbol{x}}_{\varepsilon,k}^{+T}]$.
$\boldsymbol{P}_{k-1}^+$	Previous Error Covariance (an *a posteriori* estimate)
$\boldsymbol{Q}_{v,k}$	$\boldsymbol{Q}_{v,k} = \mathrm{E}[\boldsymbol{v}_k \boldsymbol{v}_k^T]$; the covariance matrix of the additive noise $\boldsymbol{v}_k$ in the measured matrix $\boldsymbol{z}_k$.
$\boldsymbol{Q}_{w,k}$	$\boldsymbol{Q}_{w,k} = \mathrm{E}[\boldsymbol{w}_k \boldsymbol{w}_k^T]$; system model process noise covariance.
$\boldsymbol{v}_k$	Observation noise in $\boldsymbol{z}_k$ measured at interval k.
$\boldsymbol{w}_k$	Process noise at interval k.
$\boldsymbol{x}_k$	The state vector at interval k of the process.
$\hat{\boldsymbol{x}}_k^+$	*A posteriori* estimate of $\boldsymbol{x}_k$. $\hat{\boldsymbol{x}}_k^+ = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k(\boldsymbol{z}_k - \boldsymbol{C}_k\,\hat{\boldsymbol{x}}_k^-)$.
$\hat{\boldsymbol{x}}_k^-$	*A priori* estimate of $\boldsymbol{x}_k$. $\hat{\boldsymbol{x}}_k^- = \boldsymbol{A}_k\,\hat{\boldsymbol{x}}_{k-1}^+ + \boldsymbol{w}_k$.
$\hat{\boldsymbol{x}}_{\varepsilon,k}^+$	The *a posteriori* error in the estimate of $\boldsymbol{x}_k$. $\hat{\boldsymbol{x}}_{\varepsilon,k}^+ = \hat{\boldsymbol{x}}_k^+ - \boldsymbol{x}_k$.
$\hat{\boldsymbol{x}}_{\bar{\varepsilon},k}^-$	The a priori error in the estimate of $\boldsymbol{x}_k$. $\hat{\boldsymbol{x}}_{\varepsilon,k}^+ = \hat{\boldsymbol{x}}_k^- - \boldsymbol{x}_k$.
$\boldsymbol{z}_k$	System state measurement at time k: $\boldsymbol{z}_k = C_k\,\boldsymbol{x}_k + \boldsymbol{v}_k$.

3.5.2 KALMAN FILTERS FOR INERTIAL SENSOR FUSION

We have discussed the orientation representations based on Euler angles, rotation matrices, and quaternion rotation operators. We also showed the conversions matrix from one rotation operator to another. Orientation of a sensor board can be estimated and tracked as the board moves or rotates over time. An accelerometer estimates the gravity direction. A magnetometer estimates the geomagnetic field. Combining the two sensors, we are able to estimate the orientation of the

body frame (sensor board) relative to the fixed earth frame. By doing this repeatedly over short time intervals, we can measure real time rotation. We also discussed that since an accelerometer and a magnetometer can suffer from external disturbances, a gyro sensor can be fused to improve the accuracy of the orientation estimation. A gyro sensor itself can estimate orientation changes, but not absolute orientation. This problem can be resolved by fusing the gyro data with data from an accelerometer and a magnetometer.

In more practical situations, sensor data for the orientation estimation is real-time and needed to track the trajectory of the samples. Kalman filtering has been commonly used for this problem. Although MEMS and sensor technologies are constantly improving, limited power consumption and computation resources for Kalman filtering in embedded sensor systems are still the norm. There are quaternion-based orientation estimation methods that have been accepted to be stable and efficient in computations [4, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]. In this section, we review Kalman filtering schemes for orientation estimation with different combinations of sensor fusion by accelerometer, magnetometer, and gyroscope. Much more details of mathematical derivations are available in [23, 60] where the latter reference used quaternion-based Kalman filtering for various combinations of sensor fusion.

There are innumerable ways to write Kalman filter equations for inertial navigation. The figures in the following subsections are just two examples. Both filters are:

Indirect in that they operate on errors in the state variable, not the state itself. This is required, as large rotation operators are non-linear.

Complementary in that we are comparing orientation estimates from multiple sources.

Accelerometer plus magnetometer plus gyroscope

Referred to as Magnetic, Angular Rate and Gravity (MARG), this configuration offers an optimal combination of sensors for smooth tracking of orientation and separation of gravity and linear acceleration. This system is capable of yielding absolute orientation data in quaternion form with respect to magnetic north. Figure 3.22 is simplified so that we may focus on high level details. The interested reader is referred to [23] and [60] for details.

The magnetic calibration block in Fig. 3.22 was discussed in Section 2.2.1. The Indirect Kalman state variable is:

$$\boldsymbol{X}_{\varepsilon,k} = \begin{bmatrix} \boldsymbol{q}_{g\varepsilon,k} \\ \boldsymbol{q}_{m\varepsilon,k} \\ \boldsymbol{b}_{\varepsilon,k} \end{bmatrix}, \tag{3.78}$$

where:

$\boldsymbol{q}_{g\varepsilon,k}$ is the 3×1 vector component of the quaternion $q_{g\varepsilon,k}$ that models the error in orientation tilt angle relative to the true gravity vector ${}^{S}\boldsymbol{g}_k$;

$\boldsymbol{q}_{m\varepsilon,k}$ is the 3×1 vector component of the quaternion $q_{m\varepsilon,k}$ that models the error in orientation tilt angle relative to the true geomagnetic vector ${}^{S}\boldsymbol{m}_k$; and

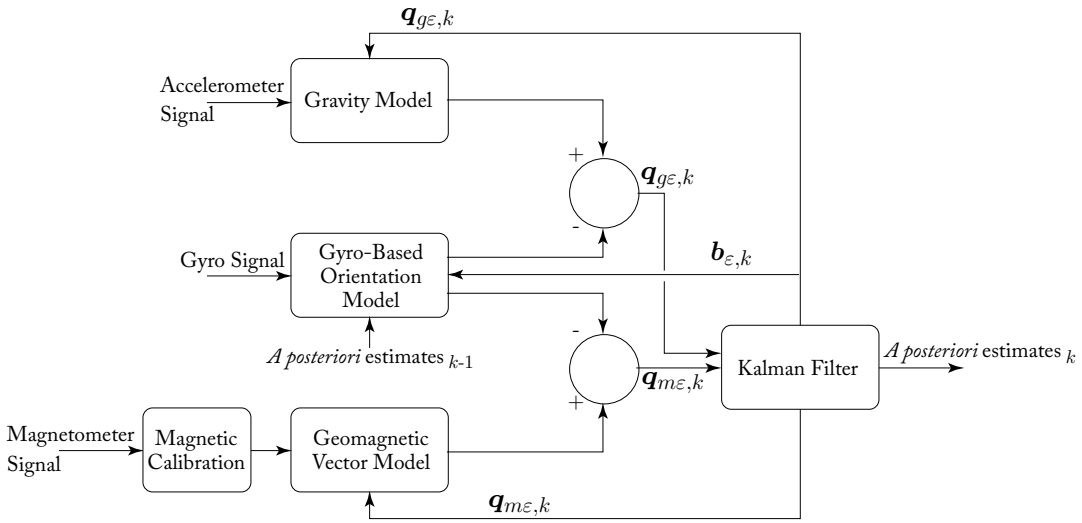

Figure 3.22: Accelerometer plus magnetometer plus gyroscope block diagram.

$\boldsymbol{b}_{\varepsilon,k}$ is the 3×1 vector component of the quaternion $b_{\varepsilon,k}$ (deg/s) which models the error in the estimate of the zero-rate gyroscope offset.

The observation model is:

$$\boldsymbol{z}_{\varepsilon,k} = \begin{pmatrix} \boldsymbol{q}_{zg\varepsilon,k} \\ \boldsymbol{q}_{zm\varepsilon,k} \end{pmatrix} = \boldsymbol{C}_k \boldsymbol{x}_{\varepsilon,k} + \boldsymbol{v}_k = \boldsymbol{C}_k \begin{pmatrix} \boldsymbol{q}_{g\varepsilon,k} \\ \boldsymbol{q}_{m\varepsilon,k} \\ \boldsymbol{b}_{\varepsilon,k} \end{pmatrix} + \begin{pmatrix} \boldsymbol{v}_{qzg,k} \\ \boldsymbol{q}_{qzm,k} \end{pmatrix}, \tag{3.79}$$

where

$\boldsymbol{z}_{\varepsilon,k}$ is the 6×1 input vector to the Kalman filter;

$\boldsymbol{q}_{zg\varepsilon,k}$ is the *a priori* error in the gyroscope-based gravity vector;

$\boldsymbol{q}_{zm\varepsilon,k}$ is the *a priori* error in the gyroscope-based geomagnetic vector;

$\boldsymbol{C}_k$ is the 6×9 measurement matrix;

$\boldsymbol{v}_{qzg,k}$ is the 3×1 gravity measurement noise vector; and

$\boldsymbol{v}_{qzm,k}$ is the 3×1 geomagnetic measurement noise vector.

This filter is optimized to compute device orientation. Once orientation is known, it is easy to compute linear acceleration, Euler angles, rotation matrices, and other related items.

Accelerometer plus Gyroscope

Figure 3.23 shows a stripped-down version of the filter from Figure 3.22, using only a gyroscope in addition to an accelerometer also yields the ability to smoothly measure rotation in 3D space, although the system can only yield orientation to the horizontal global frame of reference. In this case, the system has no sense of magnetic north. Computation of yaw is not supported. This configuration is commonly known as an Inertial Measurement Unit (IMU). This version of the filter may be applicable to game controllers, where it is not required that you know orientation relative to North.

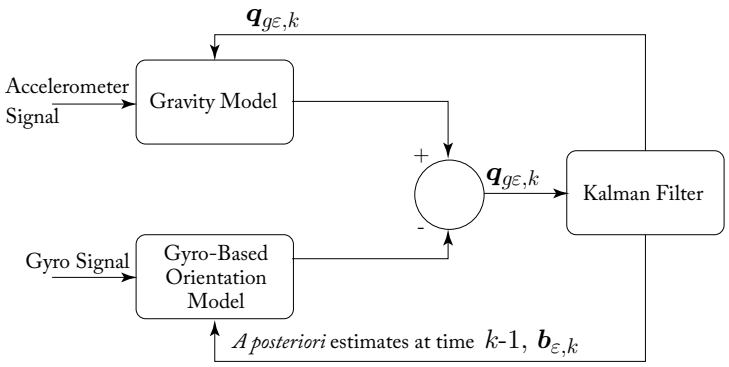

Figure 3.23: Accelerometer plus gyroscope block diagram.

3.6 TOOLS

3.6.1 NUMERICAL ANALYSIS

Sensor fusion algorithm development is often done using numerical analysis tools such as MATLAB®, Mathematica®, GNU Octave, Scilab, and others. An advantage of these environments is that vector and matrix operations are as easy to program as scalar operations. Tabular data can be easily loaded and operated on. This is seen in the virtual gyro MATLAB implementation shown in Listing 3.1. Variables `Mref`, `Aref`, and `time` are vector variables. `MagData`, `AccData`, `A`, `oldRm` and `Rm` are matrices. `load_sensor_dataset()` and `wahba()` are subroutines not shown. The former simply loads a dataset of time, accelerometer and magnetomer readings. The latter computes a rotation matrix based upon a solution to the problem originally posed by Grace Wahba [59] in 1965.

Listing 3.1: Virtual gyro pseudo-code for MATLAB

```
Mref = [0; 24.34777; -41.47411]; % reference geomagnetic
    vector
```

```
Aref = [0; 0; 1]; % reference gravity vector
[time, MagData, AccData] = load_sensor_dataset(dirName); %
   load data
deltaT = time(2)-time(1);
[NP, num_d] = size(time);
for i=1:NP % cycle through all points in the data set
    if (i==1)
       oldRm = wahba(Aref, AccData(1,:)', Mref, MagData(1,:)')
          ;
       Rm = oldRm;
    else
       [A] = wahba(AccData(i-1,:)', AccData(i,:)', MagData(i
          -1,:)', MagData(i,:)');
       Rm = A*oldRm;
    end
    OMEGA = (Rm-oldRm)*oldRm';
    current_rates(1) = (OMEGA(3,2)-OMEGA(2,3))/2;
    current_rates(2) = (OMEGA(1,3)-OMEGA(3,1))/2;
    current_rates(3) = (OMEGA(2,1)-OMEGA(1,2))/2;
    current_rates = current_rates/deltaT;

end
```

This class of tools include excellent graphics capabilities as well, making it very easy to visualize results. A number of the figures for this text were generated using MATLAB.

3.6.2 TOOLS TO CREATE FIELDED IMPLEMENTATIONS

Once an algorithm has been proved via numerical analysis, it is commonly rewritten in a language native to the hardware platform in use. For microcontrollers, this is normally in C or C++ using development environments such as IAR Embedded Workbench®, Arm® MBED™, MCUXpresso IDE, ARM®, Keil® MDK™, Atollic® TrueSTUDIO®, or GNU GCC. Android Studio is used for developing applications for Android™ platforms. Apple platforms use the Xcode development environment, which supports C, Objective-C, and Swift. Microsoft Visual Studio language support includes Basic, C++, C#, and Python.

We know of no "standard" environments specifically tailored to the development and debug of sensor fusion applications themselves. Rather, suppliers and development teams are more likely to create their own. An example is shown in Fig. 3.24. This tool, developed using Visual Studio, provides support for teams developing inertial sensor applications using the NXP's in-

ertial sensor fusion solution. It provides access to raw sensor data, magnetic and accelerometer calibration data, Kalman filter variables, etc.

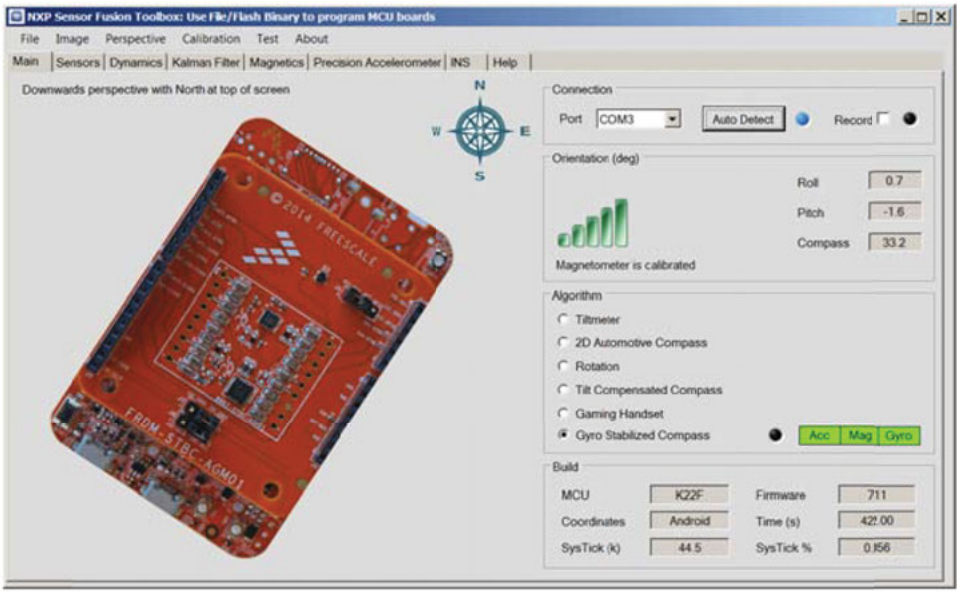

Figure 3.24: The NXP sensor fusion toolbox for Windows.

CHAPTER 4

Machine Learning for Sensor Data

4.1 INTRODUCTION

IoT applications typically include sensors and communication hardware. They are often associated with large volumes of sensor data that characterize natural phenomena and/or other physical quantities. Most include requirements for processing data to extract information that may be monetized or acted on in some way. This chapter explores the use of machine learning techniques to extract that information [155].

Machine learning can be broadly broken out into subclasses, as shown in Fig. 4.1. *Supervised learning* takes advantage of situations in which labeled datasets are available for training. It can be further divided into classification and regression techniques. *Unsupervised learning* does not rely on labeled data for training. It looks for patterns in the input data, but without any knowledge of what those patterns represent. Reinforcement learning relies on external positive or negative feedback of prior system outputs to optimize the system model going forward. We will discuss supervised and unsupervised learning in later subsections. See [167, 168] for additional information on the topic of reinforcement learning.

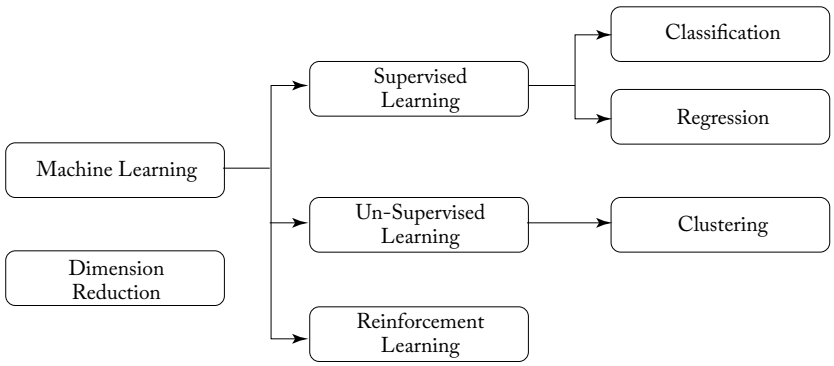

Figure 4.1: General hierarchy of machine learning techniques.

Closely aligned with machine learning techniques is the topic of dimension reduction. This set of techniques allows us to simplify the problem by either selecting a relevant subset of

input features, or recasting the input data into a lower dimensional space which retains much of the original information.

The generalized workflow for supervised and unsupervised machine learning is shown in Fig. 4.2. It begins with sensor data fed into a preprocessing stage that typically segments and denoises the data. This is followed by feature extraction, where the processed data is further reduced into features which describe the incoming data stream in some way. Examples of features would be computation of standard deviation, zero crossings, spectral components or min/max values. The data stream is then split into portions used for learning and model validation, as well as unknown data sent to the final computed model for classification, regression, or clustering.

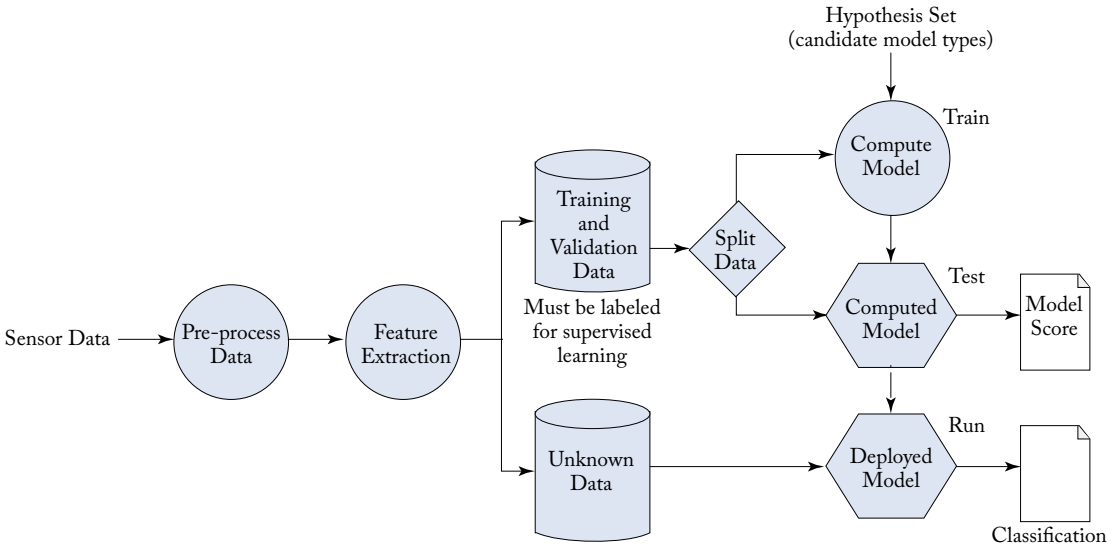

Figure 4.2: General workflow for supervised and unsupervised machine learning.

In this chapter, we explore various attributes of sensor data. We show how sensor data is typically processed and how to use data and features to learn about and detect events of interest. Finally, we compare cloud-based and embedded machine learning systems.

4.2 SENSOR DATA ACQUISITION

4.2.1 STRUCTURED VS. UN-STRUCTURED DATA

Data types can be categorized as structured or unstructured. Structured data can be visualized as rows and columns of a matrix with numerical values which can be efficiently stored in database format. Unstructured data types include text, people and objects within images, and social media data. Inertial sensor data is a good example of the structured data type. It is often generated and processed in real time.

Sensors are often so small that they can be embedded in personal devices such as smartphones. With low power consumption, "always-on" operation is possible, and devices can continuously generate new data. Continuous streams of sensor data can tax storage systems and network bandwidth because of the huge volume of data being generated. Machine learning techniques can compress and interpret data prior to transmission, conserving system bandwidth and storage in the process.

4.2.2 DATA QUALITY

Data samples can be lost or corrupted during transmission. If the learning process from sensor data requires structured data with no missing values, the transmission protocol must include error checking and data recovery features. When data is collected, unexpected outliers and/or measurement noise can occur. There are many studies that discuss removal of outliers and random measurement noise, as well as missing data recovery [61, 62]. There are also various adaptive filtering schemes to improve the quality of real-time sensor data [63, 64, 65, 66, 156].

Another issue effecting data quality is sensor sampling rate vs. data resolution. The two often operate at cross purposes, and tradeoffs may impact machine learning performance. In IoT applications where devices are connected in a network, there may be trade-offs due to limitations in communication capability.

4.2.3 INHERENT VARIABILITY

In many applications, sensor data is used for monitoring machinery or human activity where the data statistics are non-stationary [67, 68]. In other words, statistics of the measurement data have time-varying properties. Variations may depend on when or where the sensor data are observed, and might be considered variable "bias" that needs to be removed from data prior to machine learning. This is also known as "concept drift." Alternately, time-based variations might convey valuable information which we hope to extract. We may require machine learning algorithms [155] to relearn or adapt to "bias variations" over time. In many applications, time-based variation in sensor data must be analyzed before a machine learning algorithm is performed.

4.3 FEATURE EXTRACTION

Many machine learning algorithms require features extracted from sensor data as input. There are various signal processing techniques that can be applied for feature extraction. We categorize those techniques into time domain, frequency domain, and time-frequency domain features. Dimension reduction techniques are introduced in sections that follow, since raw sensor data collected from multiple sensors typically consist of multi-dimensional signals and may include redundant dimensionality. Feature selection methods are also briefly described.

4.3.1 TIME-DOMAIN FEATURES

Raw sensor data is often collected as time-series in real time. Time domain features extracted via traditional descriptive statistics include: mean, variance, skewness, kurtosis, zero-crossing rate, crest factor, and so on. In the following, we summarize the definitions of various feature types. Let $x_t \in \mathbb{R}$ be a scalar measurement data at time t. Let N be the segment size of samples to calculate features

$$\text{(Mean)} \qquad \frac{1}{N}\sum_{t=1}^{N} x_t. \tag{4.1}$$

$$\text{(Variance)} \qquad \frac{1}{N-1}\sum_{t=1}^{N}(x_t-\bar{x})^2, \tag{4.2}$$

where $\bar{x}$ is the sample mean

$$\text{(Skewness)} \qquad \frac{\frac{1}{N}\sum_{t=1}^{N}(x_t-\bar{x})^3}{\left[\frac{1}{N-1}\sum_{t=1}^{N}(x_t-\bar{x})^2\right]^{1.5}}. \tag{4.3}$$

$$\text{(Kurtosis)} \qquad \frac{\frac{1}{N}\sum_{t=1}^{N}(x_t-\bar{x})^4}{\left[\frac{1}{N}\sum_{t=1}^{N}(x_t-\bar{x})^2\right]^{2}} \tag{4.4}$$

$$\text{(Zero-crossing rate)} \qquad \frac{1}{N}\sum_{t=2}^{N}\mathbf{1}(x_t x_{t-1}), \tag{4.5}$$

where $\mathbf{1}(x)$ is an indicator function, 1 if $\text{sign}(x_n) \neq \text{sign}(x_{n-1})$, 0 otherwise.

$$\text{(Crest factor)} \qquad \frac{\max(x_t)}{x_{rms}}. \tag{4.6}$$

4.3.2 FREQUENCY-DOMAIN FEATURES

Frequency-specific information is often invisible in time domain data. The Fast Fourier transform (FFT) is broadly used to convert signals into the frequency domain. It is an algorithm that computes the discrete Fourier transform (DFT) efficiently [156]. The frequency components in a DFT reveal periodicity of time-series raw sensor data. In this context, the FFT is a useful tool for monitoring applications with rotating devices [1, 69, 70, 71, 72].

There are more sophisticated tools for frequency domain feature extraction. The power cepstrum is defined as the inverse Fourier transform of logarithm of the frequency spectrum. Bi-spectrum and Tri-spectrum are examples of high-order statistics spectrum techniques [73, 74], whereas power spectrum is based on the second-order statistics. The methods above are non-parametric approaches. There are parametric methods as well. For those who are interested in learning more about frequency-domain features, see [75] and [76].

4.3.3 TIME-FREQUENCY FEATURES

The frequency domain features described in Section 4.3.2 are typically not able to deal with non-stationary data. There are time-frequency domain analysis tools that can reveal features even in the case of non-stationary statistics. The most straightforward approach is short-time Fourier transform (STFT) which is computed by Fourier transform of segmented data after dividing the entire time-series into multiple segments [75].

Wavelets, commonly used for compression and denoising functions, can also be used to compute time-frequency features for machine learning purposes. A single-level wavelet transform decomposes an input waveform into two sub-signals of half the original waveform length. Figure 4.3 illustrates implementation of a wavelet analysis process using decimation filters. This specific process is used to implement a four-level wavelet transform. We start by applying high- and low-pass decimating filters to an input signal $\boldsymbol{s}_1$ to obtain $\boldsymbol{a}_1$ and $\boldsymbol{d}_1$. We then repeat the process, using $\boldsymbol{a}_n$ as $\boldsymbol{s}_{n+1}$ at each new stage of decomposition. At each stage, the input signal is recast as the sum of a trend signal $\boldsymbol{a}_n$ and a difference signal $\boldsymbol{d}_n$. The original signal can be easily reconstructed using a synthesis filter bank. Quadrature mirror filter banks (QMF) are duals of wavelet transforms [160, 161] and their theory is discussed in [158].

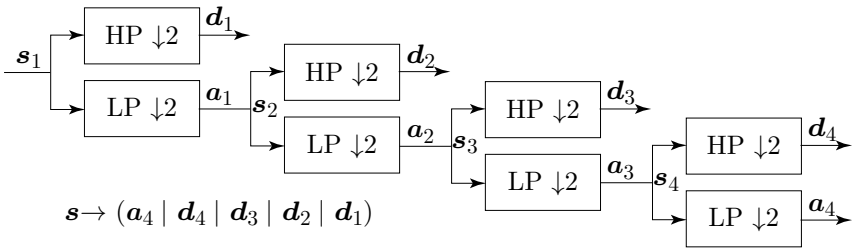

Figure 4.3: Four-level discrete wavelet transform decomposition using decimation filters.

The technique is referred to as wavelets because each difference signal $\boldsymbol{d}_m$ is implemented via a dot product operation:

$$\boldsymbol{d}_n = \boldsymbol{s}_n \cdot \boldsymbol{W}_n, \tag{4.7}$$

where the coefficients of wavelet $\boldsymbol{W}_n$ describe a waveform of limited duration and average value of zero. There are many families of wavelets, each of which is optimal for specific classes of problems [77, 78, 79].

Elements of the trend outputs are derived in a comparable manner:

$$\boldsymbol{a}_n = \boldsymbol{s}_n \cdot \boldsymbol{V}_n, \tag{4.8}$$

where $\boldsymbol{V}_n$ is referred to as a scaling signal. $\boldsymbol{V}_n$ is selected so that power is conserved at each stage of the transformation.

Figure 4.4 shows the wavelet decomposition for a simple signal plus noise. Notice how $\boldsymbol{a}_4$ and $\boldsymbol{d}_4$ have significantly less resolution in the time axis than does the original signal.

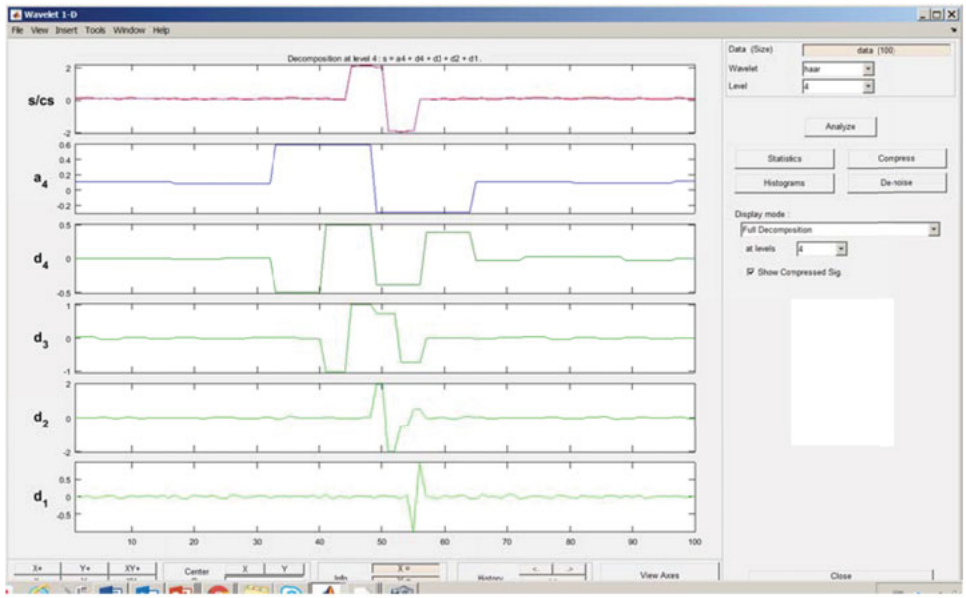

Figure 4.4: Wavelet-compressed signal shown in MATLAB wavelet toolbox.

4.3.4 DIMENSION REDUCTION

When many sensors are present in a system, they often produce redundant information, and we may want to reduce the dimensionality of the data. Principal component analysis (PCA) is a statistical analysis tool to reduce the dimensionality of collected sensor data [80, 81]. PCA maps the multi-dimensional data into the eigenvector space and projects the original data onto eigenvectors that correspond to large eigenvalues. The idea of PCA is to transform and represent the data with fewer principal components (axes or variables) than used in the original data representation. There are variations of this idea. Independent component analysis (ICA) extracts statistically independent components [82]. Kernel PCA [83] is an extension of PCA which uses kernel methods to handle nonlinear datasets. See Section 4.4.3 for an introduction to kernel methods.

4.3.5 FEATURE SELECTION

The determination of features to be used to train machine models is often application specific. In speech processing, we may use cepstral parameters [84] because of their association with the vocal tract. In image processing, the Discrete Cosine Transform (DCT) is used in JPEG, and in JPEG 2000 [159] the wavelet transform is used because of its multiresolution and data compaction capability. The Karhunen Loéve transform (KLT) [162], which is signal-specific, is also used in compression applications. We note that the KLT relates to eigenvalues and eigenvectors

of the covariance matrix of the data and hence the KLT also relates to principal components which were discussed in the previous section and are revisited later in this chapter.

Feature selection can be done by manually, or via algorithms such as genetic algorithms [85], differential evolution [86], or simulated annealing [87]. Refer to [88] for additional details.

4.4 SUPERVISED LEARNING

In Section 4.1, we briefly noted two sub-classes of supervised machine learning: regression and classification. Both estimate a function f, derived from pairs of observation data and known output response $(\boldsymbol{x}(0), \boldsymbol{x}(1), \ldots \boldsymbol{x}(n); y(0), y(1), \ldots y(n))$:

$$y = f(\boldsymbol{x}). \tag{4.9}$$

In regression problems, y is numeric. In classification problems, it is either a class assignment, or the numeric likelihood(s) or probability for specific class membership(s).

As an example, consider simple linear relationship. Learning a model is in this case equivalent to estimating the coefficients ω_1 and ω_0 in the equation:

$$y = \omega_1 x + \omega_0, \tag{4.10}$$

where x is a new measurement and y is the prediction result. The optimal choice of coefficients will minimize the residual errors, as shown in Fig. 4.5. Once trained, this model can predict output responses corresponding to new input data. The same concept of minimizing residual errors is easily extended to multidimensional problems.

Now let us consider a classification problem. Assume that we want to learn a decision boundary that separates two different classes of data samples in 2D space where each known data sample is denoted as $\boldsymbol{x} = [x_1 x_2; < \mathit{className} >]^T$ as shown in Fig. 4.6. If the classes are separable as shown, then a boundary can be found that, given new values of x_1 and x_2, can predict the associated class name for each sample ("C_{-1}" and "C_1" in the figure). There are several algorithms to handle this problem definition, including extensions for nonlinear and for cases where the pattern might not be evident to the casual observer.

4.4.1 LINEAR DISCRIMINANT ANALYSIS

Linear discriminant analysis (LDA) [89] is used both for feature reduction and as a classification scheme. It projects data points onto a vector $\boldsymbol{\omega}$ which is chosen to maximize the ratio:

$$\frac{\text{variance between classes}}{\text{variance within a class}}. \tag{4.11}$$

Consider the example of Fig. 4.7. We have N data vector samples $\boldsymbol{X} = [\boldsymbol{x}(0), \boldsymbol{x}(1), \ldots, \boldsymbol{x}(N-1)]$. N_{-1} of N samples belong to class $-1(C_{-1})$ and N_1 belong to class $1(C_1)$,

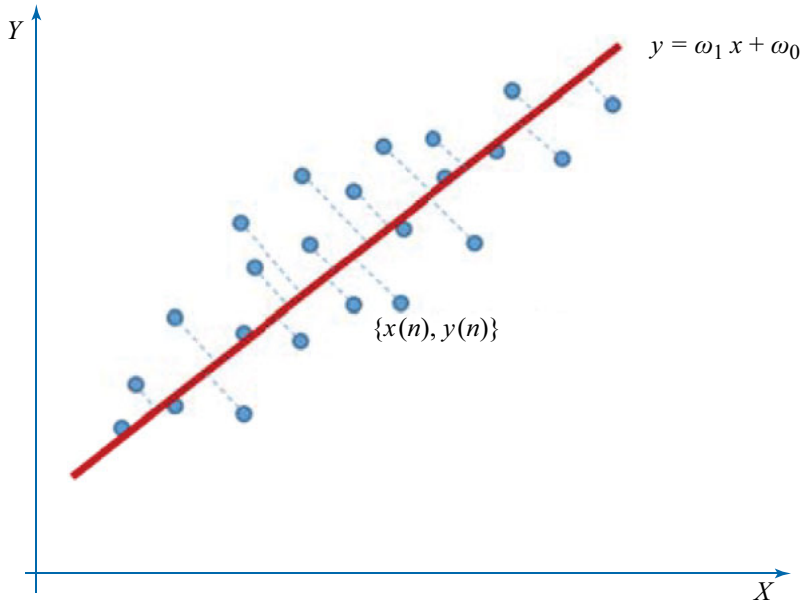

Figure 4.5: Linear regression in 2D space.

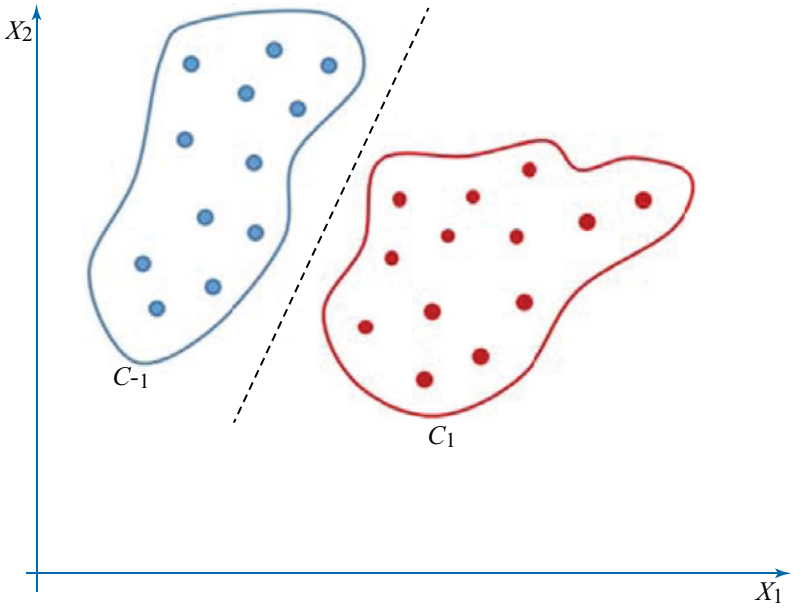

Figure 4.6: Linear classification boundary in 2D space.

where $N_{-1} + N_1 = N$. The mean vector of each class is defined as

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n \in C_k} \boldsymbol{x}(n), \tag{4.12}$$

where $k = -1, 1$. The sample means that are projected onto the vector $\boldsymbol{\omega}$ are given by $\hat{\boldsymbol{\mu}}_k = \boldsymbol{\omega}^T \boldsymbol{\mu}_k$ for $k = -1, 1$. We want to maximize the distance between the projected means, which is given by

$$\hat{\boldsymbol{\mu}}_1 - \hat{\boldsymbol{\mu}}_{-1} = \boldsymbol{\omega}^T (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1}). \tag{4.13}$$

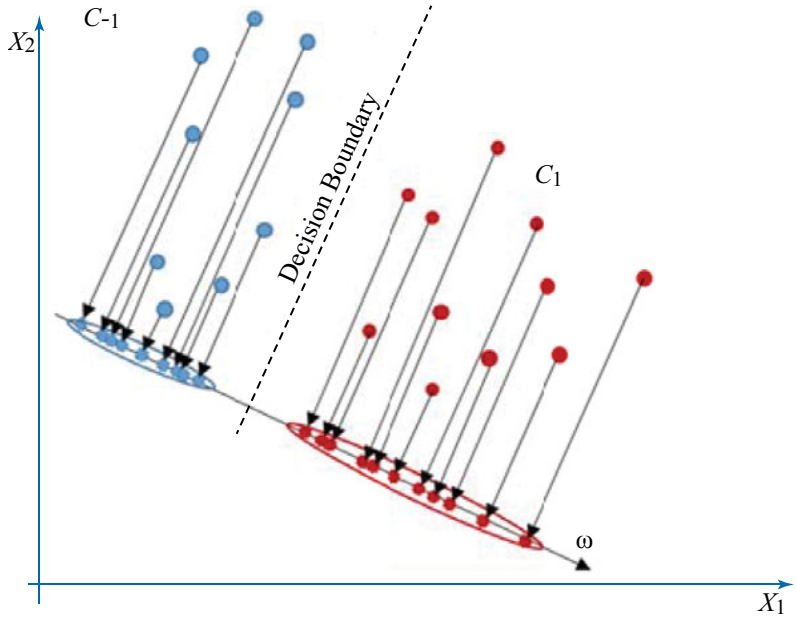

Figure 4.7: An example of linear discriminant analysis (LDA) that finds a decision boundary with its orthogonal vector $\boldsymbol{\omega}$.

Suppose also covariance of each class is denoted as $\boldsymbol{\Sigma}_{-1}$ and $\boldsymbol{\Sigma}_1$, respectively. The covariances of the projected samples are $\boldsymbol{\omega}^T \boldsymbol{\Sigma}_{-1} \boldsymbol{\omega}$ and $\boldsymbol{\omega}^T \boldsymbol{\Sigma}_1 \boldsymbol{\omega}$. The object is to maximize the following function:

$$J(\boldsymbol{\omega}) = \frac{\boldsymbol{\omega}^T (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1})(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_{-1})^T \boldsymbol{\omega}}{\boldsymbol{\omega}^T (\boldsymbol{\Sigma}_{-1} + \boldsymbol{\Sigma}_1) \boldsymbol{\omega}}. \tag{4.14}$$

Solving the optimization problem results in the vector $\boldsymbol{\omega}$ that maximizes $J(\boldsymbol{\omega})$.

Once we know $\boldsymbol{\omega}$, and given the probabilities of C_{-1} and C_1, we can classify a new point $\boldsymbol{x}$ to C_{-1} if

$$\omega^T \left[x - \left(\frac{\mu_1 + \mu_{-1}}{2} \right) \right] > \log \frac{p(C_{-1})}{p(C_1)} \tag{4.15}$$

and to C_1 otherwise.

4.4.2 SUPPORT VECTOR MACHINES

One strategy for linear classification is to maximize the distance between classes. The motivation for using Support Vector Machines (SVMs) is also to maximize the margins between two classes. Linear SVMs determine the classifier that maximizes the margin between two classes, as illustrated in Fig. 4.8.

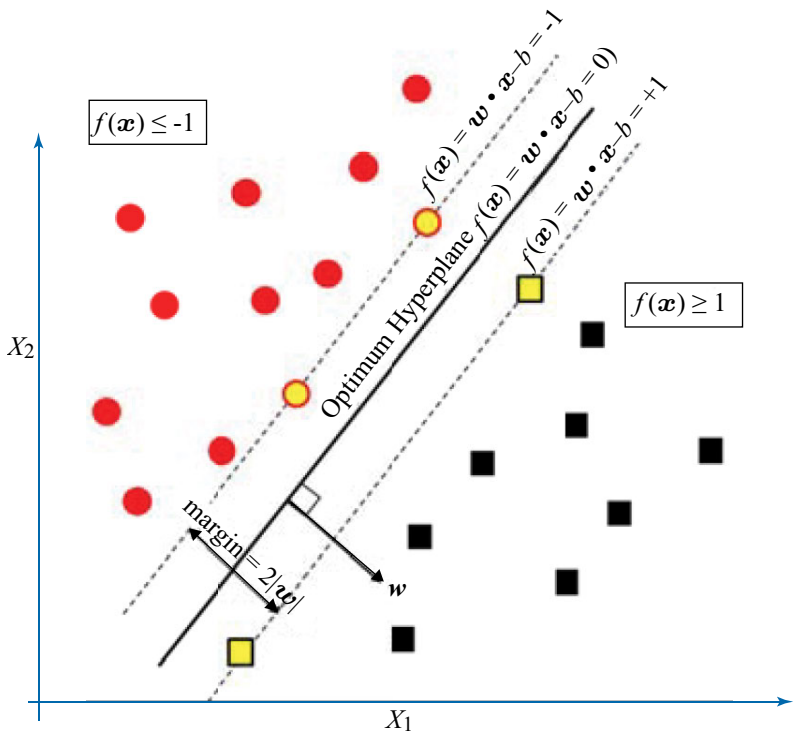

Figure 4.8: Linear support vector machine (SVM).

4.4.3 KERNEL FUNCTIONS

The data samples in the left side of Fig. 4.9 are separableo but not by a linear classifier. This can be solved by using nonlinear functions to map the data into higher-dimensional feature spaces.

We can then construct linear hyperplanes to separate the data classes. This is shown on the right side of Fig. 4.9.

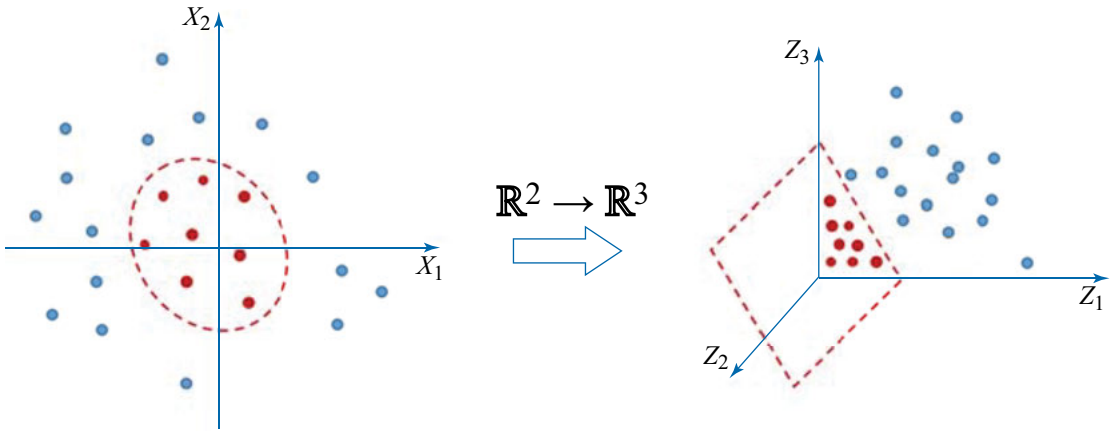

Figure 4.9: Mapping to higher dimension.

Let us define the mapping function as:

$$\boldsymbol{x} \mapsto \Phi(\boldsymbol{x}). \tag{4.16}$$

In the higher dimensional space we need to learn $\boldsymbol{\omega}$ and b of the classifier

$$f(\boldsymbol{x}) = \boldsymbol{\omega} \cdot \Phi(\boldsymbol{x}) + b. \tag{4.17}$$

The Representer Theorem [90] allows us to define $\boldsymbol{\omega}$ as

$$\boldsymbol{\omega} = \sum_{n=1}^{m} \alpha_n \Phi(\boldsymbol{x}_n) \tag{4.18}$$

for some variables α_n where $n = 1, \ldots, m$. By substituting Eq. (4.18) into (4.17), the decision model can be given by

$$f(\boldsymbol{x}) = \sum_{n=1}^{m} \alpha_n \Phi(\boldsymbol{x}_n) \cdot \Phi(\boldsymbol{x}) + b. \tag{4.19}$$

An explicit mapping into higher dimensions can be avoided by noting that Eq. (4.19) depends only on the inner product $\Phi(\boldsymbol{x}_n) \cdot \Phi(\boldsymbol{x})$. We can often obtain explicit expressions for that inner product that are more efficient than doing the explicit mapping followed by dot product operator.

Using the "kernel trick,"[1] which specifies kernel $\kappa(\boldsymbol{x}_n, \boldsymbol{x}) = \Phi(\boldsymbol{x}_n) \cdot \Phi(\boldsymbol{x})$, we can redefine Eq. (4.19) as

$$f(\boldsymbol{x}) = \sum_{n=1}^{m} \alpha_n \kappa(\boldsymbol{x}_n, \boldsymbol{x}) + b. \tag{4.20}$$

[1] "Kernel Trick" is actually the formal name used for this technique.

Then the goal is to find the coefficients $\{\alpha_n\}_{n=1}^m$ and the corresponding support vectors $\{\boldsymbol{x}_n\}_{n=1}^m$ via an optimization process [27]. There are several conditions for the kernel κ, but instead of going through the mathematical details of the conditions, we summarize a few examples of commonly used kernels in Table 4.1. Figure 4.10 illustrates an example of and SVM classifier with the mapping function of Eq. (4.16). Without explicitly defining $\Phi(\cdot)$, SVMs uses the kernel functions $\kappa(\cdot,\cdot)$ to find a classifier in higher-dimensional feature space.

Table 4.1: Typical kernel examples

Kernels	Definitions
Gaussian radial basis	$\kappa(\boldsymbol{x},\boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|}{2\sigma^2}\right)$ where the kernel size $\sigma > 0$.
Polynomial	$\kappa(\boldsymbol{x},\boldsymbol{x}') = (\boldsymbol{x}^T\boldsymbol{x}' + c)^d$ where $d \geq 0$.
Sigmoid	$\kappa(\boldsymbol{x},\boldsymbol{x}') = \tanh(\eta\,\boldsymbol{x}^T\boldsymbol{x}' + \theta)$.

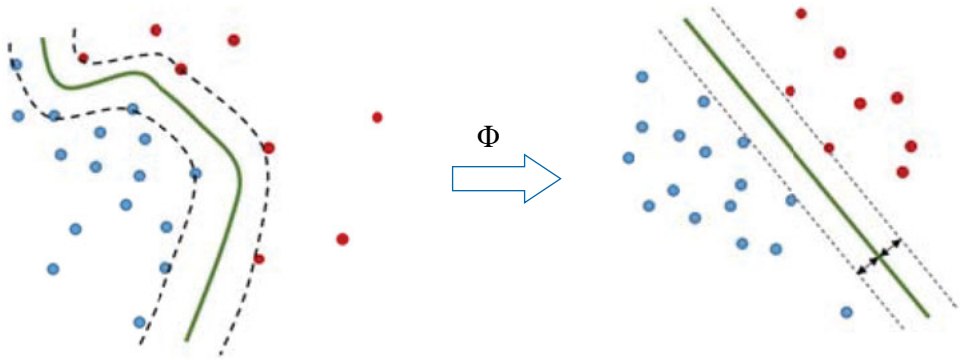

Figure 4.10: SVM classifier in higher dimensional feature space.

We have briefly summarized several aspects of supervised learning with very limited algorithm explanations. Those who are interested in more extensive contents are recommended to read machine learning-related books in the literature [27, 28, 89].

4.5 UNSUPERVISED LEARNING

Unsupervised learning is typically more challenging than supervised learning in that the former tends to be subjective, whereas the latter usually has simple goals such as prediction and classification. Also, unsupervised learning has access only to observed input data but does not have any information regarding what would be the desired result from the data. In this context, unsupervised learning often tends to infer a function that describes intrinsic structure of data.

In the following, we introduce some examples of unsupervised learning algorithms. Principal component analysis (PCA) [162, 163] is a traditional method to derive low-dimensional representation from high-dimensional dataset. The idea of PCA is to linearly transform the data in to a principal component subspace such that highly variable data are aligned on the first principal component. In order to reduce the dimensionality, the algorithm searches for the subspace structure that can almost equivalently represent the given dataset. This can be mathematically solved by singular value decomposition [163] which relates to eigen analysis. The procedure is straightforward and details are available in signal processing books or online materials. Another example of unsupervised learning is clustering analysis. Clustering is a task to group similar data in a cluster under a certain criterion. There are several approaches for clustering such as k-means [91] and Gaussian mixture model (GMM). Vector quantization [91, 92, 93, 94, 95] typically uses k-means algorithm to determine a codebook for signal compression. When is used for clustering VQ represents sensor data by centroids of the formed cells. The k means algorithm can be used for training in machine learning algorithms. Unsupervised learning is also closely related to density estimation problems in that the density estimator finds a distribution structure of the given dataset.

4.6 REMARKS—LEARNING FROM SENSOR DATA

We have briefly described machine learning algorithms for building decision-making models. Learning from sensor data can be done using both supervised and unsupervised learning techniques. When sensors provide multi-dimensional data, we may want to determine principal components of the data axes that can be used to represent the same information with lowered dimensionality. Features extracted from raw sensor data provide good data sources for machine learning of models to differentiate various conditions and activities of people and machinery. Even without data labels, we can search for patterns in the data using clustering techniques. Unsupervised learning can also be used for anomaly detection, where we look for system outputs inconsistent with past patterns in the data.

4.7 PERFORMANCE EVALUATION

Classification problems are evaluated via confusion matrices constructed based upon known datasets applied to a learned model. They are structured such that actual classifications are aligned along one axis of the matrix, and predicted classifications are aligned along the other. Each test pattern is assigned to a matrix entry based upon its actual and predicted classification. Ideally all test cases lay on the diagonal in Table 4.2. However, machine learning is a statistical process, and there will almost always be off-diagonal components. In machine condition monitoring (MCM) applications, false negatives result in missed opportunities to service equipment before it becomes a critical issue. Standard metrics derived from the confusion matrix are as follows.

Accuracy: 1 – error = (TP + TN)/(PP + NP). Probability of a correct classification.

Sensitivity: TP/PP. The ability of the test to detect a positive element from a positive population.

Specificity: TN/NP. The ability of the test to correctly rule out a condition from a condition free population.

Table 4.2: Confusion matrix

Actual	Predicted Positive	Predicted Negative
PP = TP + FN Positive Population	TP = True Positive	FN = False Negative
NP = FP + TN Negative Population	FP = False Positive	TN = True Negative

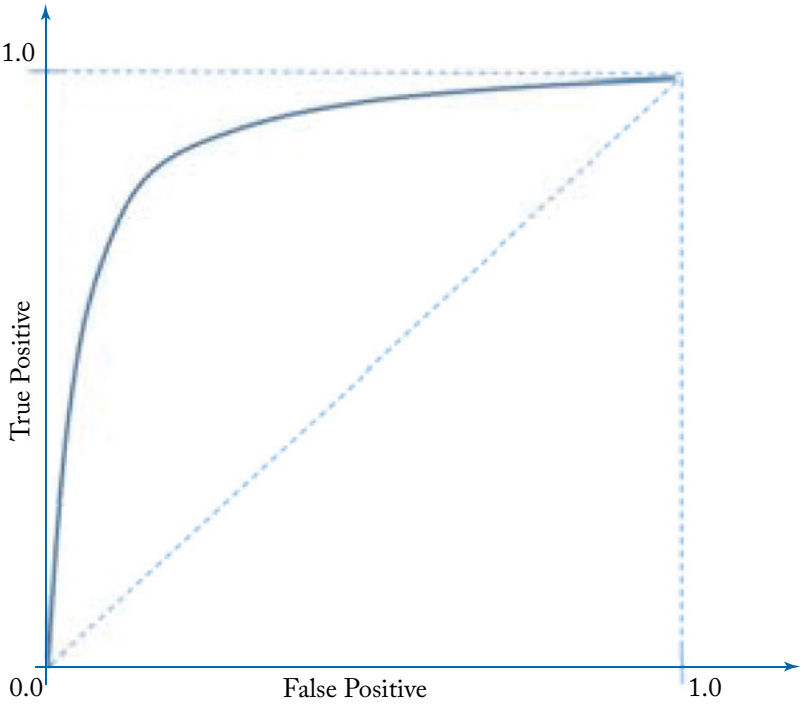

Figure 4.11: Receiver operating characteristic (ROC) curve.

The Receiver Operating Characteristic (ROC) curve plots TP against FP at different levels of threshold settings. An example of ROC curve is illustrated in Fig. 4.11. ROC curves provide a tool for evaluating machine learning algorithms. The area under the curve is often used for model comparison. Larger area under the curve represents better performance in that true positives dominate over false positives.

Evaluation of unsupervised learning algorithms is hard to clearly define because algorithms are subjective and the evaluation depends on applications. But there are some kinds for validity indices such as external and internal indices. The external method is based on comparison between known clustering structure and the clustering result [96]. The internal method is based on the goodness of a clustering structure [97].

For sensor data analytics evaluation, we may have to measure computation capabilities because sensor data are generated and evaluated on embedded devices. Also, communication rate and reliability between the embedded devices and IoT center are important metrics.

4.8 DEEP LEARNING

Deep learning, which typically uses neural network architectures, is a subtype of machine learning [144, 145, 146, 147]. Neural networks have been broadly used in many research fields and they achieve good performance in many applications.

The term "deep" usually refers to much larger number of hidden layers in the neural network, whereas traditional neural networks typically have only a few layers [98, 99, 100, 101, 102]. Figure 4.12 illustrates a neural network architecture with multiple hidden layers. Each layer takes the output of the previous layer with nonlinear mapping and feed the results to the next layer. While research of multilayer neural networks with hidden layers started decades ago, deep learning became popular only recently [103] because of compelling classification results [170, 171]. There are two main issues to achieving impressive results of deep learning in applications. Deep learning requires computation power, which is supplied by high-performance GPUs. Also, deep learning needs large quantities of labeled data for training, which are provided by cloud-based systems. Convolutional neural networks (CNNs) [171] are the most popular deep learning technique used for image data and recurrent neural networks (RNNs) are used for sequential data.

A brief comparison between deep learning and traditional machine learning in described in Table 4.3. One can decide which approach is required for his/her own applications.

4.9 INTEGRATION POINT OF MACHINE LEARNING ALGORITHMS

For sensor data analytics in IoT applications, machine learning algorithms play a crucial role. Where the algorithms are executed is a very important issue because each choice comes with various advantages and disadvantages. The easiest approach for sensor data analytics is to connect all sensor devices to an IoT data center (or cloud) so machine learning algorithms are conducted

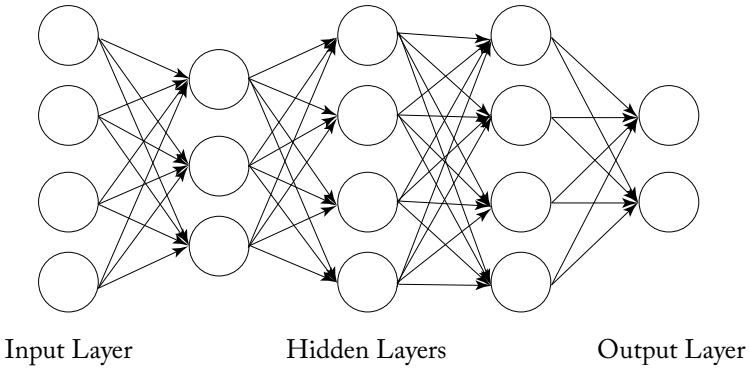

Figure 4.12: An example of a neural network structure with three hidden layers.

Table 4.3: Traditional machine learning vs. deep learning

	Traditional Machine Learning	Deep Learning
Computation (training) time	Short	Long
Training dataset	Small	Large
Manual feature extraction	Yes	No
How it works	Various automated algorithms learn model funtions and predict or classify from data	Uses neural networks that process through many layers and find relevant data features

by cloud-based computing resources. Virtually unlimited compute power makes this type of framework fast and efficient for building the statistical models. However, as the size of real-time sensor data increases exponentially, more computing (bandwidth, storage, and CPU) resources will be required over time, which drive cost up.

Another important issue we must address is privacy and security [104, 105]. Centralized sensor data analytics may be vulnerable to privacy issues as raw sensor data streams are transferred to an IoT center. To resolve the computation burden issue, distributed computing mechanisms have been developed. But still there is a room for improvement in privacy and data transmission issues between local sensor devices and distributed computing servers.

Another methodology is to push sensor data analytics and machine learning algorithms into IoT leaf nodes. As microcontrollers and embedded technologies develop, embedding ML models in an embedded sensor system itself is possible [26]. This method has advantages in that private raw sensor data streams are not necessarily transferred to the cloud and data traffic between sensors and IoT server can be significantly reduced. The computation is fully distributed, and this approach is preferred for real-time IoT applications. However, limited com-

puting power and storage at the leaf node means it is still difficult to conduct highly complicated machine learning algorithms *in situ*.

Hybrid methods are also possible. Embedded sensor systems predict, classify, and make decisions based on real-time sensor data stream. However, machine learning algorithms are conducted at an IoT center because the learning process requires high computation resources. After training a model based on a set of sensor data transferred from embedded sensor systems, the model is sent back to the embedded system for making decisions in real time. Figure 4.13 illustrates the three different integration points of machine learning algorithms.

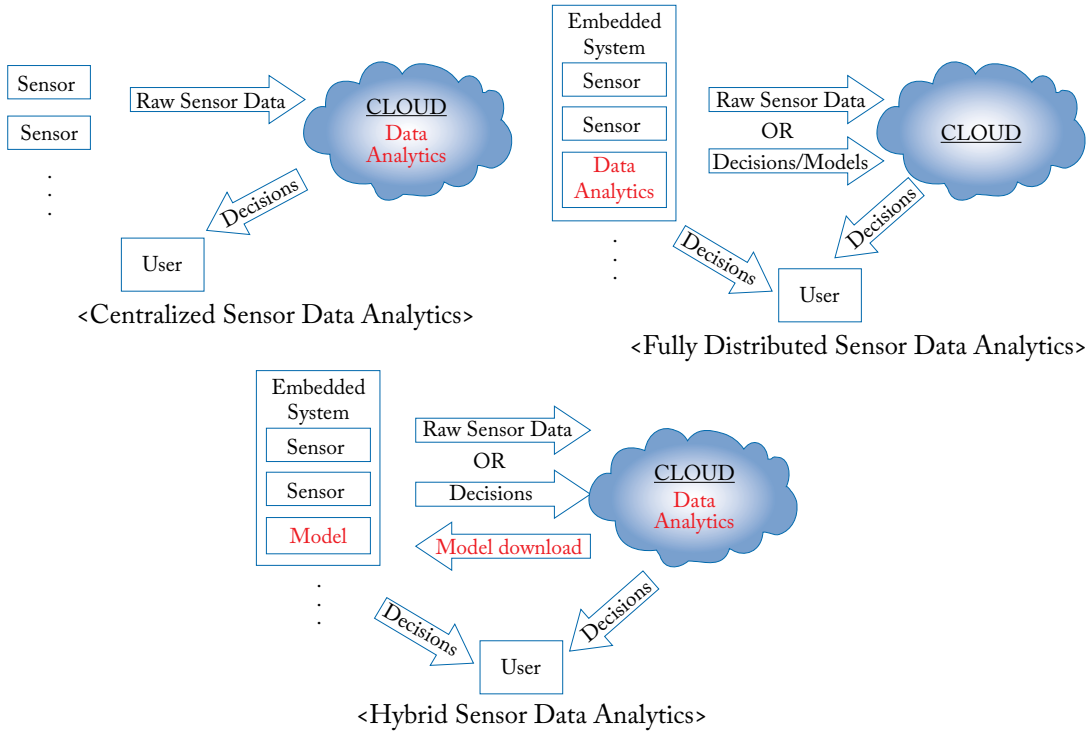

Figure 4.13: Integration points (centralized/fully distributed/hybrid) of sensor data analytics.

4.10 TOOLS FOR MACHINE LEARNING

Recent years have seen an explosion in the practical application of machine learning, resulting in an abundance of tools available to expert and novice practitioners alike. The numerical analysis tools listed in Section 3.6.1 are also applicable to the machine learning space. They include library functions for a variety of machine learning techniques. For instance, MATLAB can compute a support vector machine model with the single statement:

```
mdl1 = fitcsvm(X,Y,'KernelFunction', 'linear', 'PolynomialOrder', []);
```

Using this approach, the developer is still responsible for data import, conditioning, and plotting of results. This is a time-consuming process. Responding to that problem, Mathworks [106] introduced the *Classification Learner* app for MATLAB. It takes advantage of the fact that many supervised learning problems use tabular inputs to drive the learning process. Data tables can be quickly loaded, feature spaces explored, and multiple machine learning model types applied with just a few button clicks (Fig. 4.14).

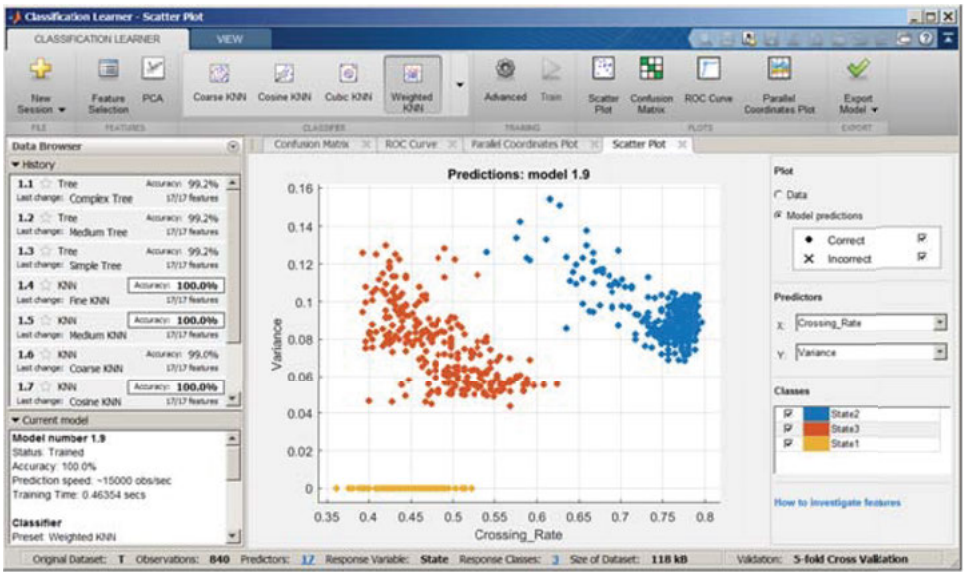

Figure 4.14: MATLAB's classification learner application.

The Python [107] and R [108] programming languages have both earned followings in the machine learning community. Machine learning applications are supported via optional plugin modules. Reference [109] suggests a set of R packages for machine learning. These include MICE [110], RPART [111], PARTY [112], CARET [113, 114], and NNET [115]. The scikit-learn [116, 117] package for Python includes support for classification, regression and clustering.

The KNIME®Analytics Platform [118] models ML tasks as graphical workflows (Fig. 4.15). It uses graphical markers to represent data import, data conditioning, machine learning, validation and reporting. Work flows can be incrementally tested as they are built, and exported for use on other platforms. KNIME is available for Windows, Linux and Mac OSX platforms, as well as the Microsoft Azure Cloud Platform.

Other tools supporting graphical modeling of workflows include IBM® SPSS® Modeler [119] and Microsoft Azure Machine Learning (ML) [120]. SPSS Modeler is available in Windows and cloud versions. Azure ML is cloud-based only. An example Azure ML workflow

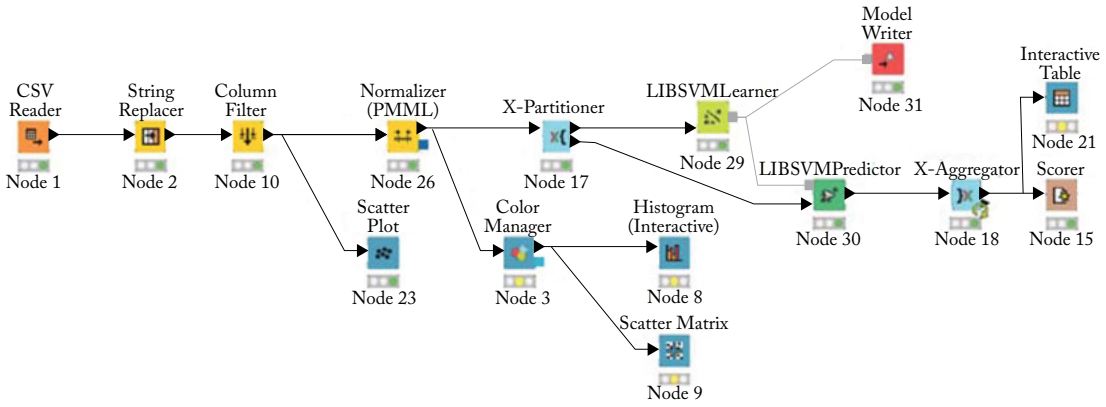

Figure 4.15: A sample KNIME workflow.

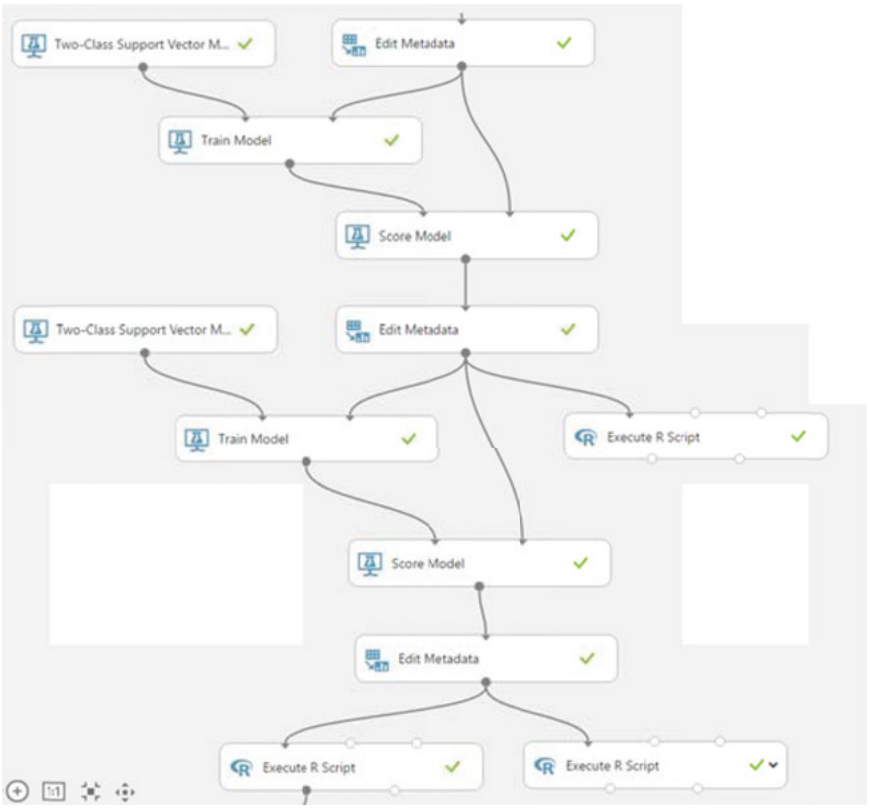

Figure 4.16: A sample Azure machine learning workflow.

is shown in Fig. 4.16. Each node in the figure represents a specific operation type, and operation parameters are assigned in a properties panel in the Azure ML web-based interface. Notice the nodes labeled "Execute R Script." Azure supports integration of custom sub-functions written in R. Example functions might include adding a graphical output not otherwise supported by Azure ML, or implementing custom algorithms.

TensorFlow™ [121], originally developed at Google, is now an open source library for numerical computation and deep learning. It continues to be used by the Alphabet companies and is available as part of Google's Cloud Machine Learning Engine, as well as Windows, Mac OS X, and Linux. It models problems as computational graphs composed of a series of nodes, each of which consume zero or more and generate at least one tensor output.

The tools listed above are but a small sampling of those available for research and development. Many are free and/or open source. Several are cloud based. Some are hosted on PCs, Macs, and Linux. At the other end of the spectrum, the authors have implemented Gaussian Mixture Models and Support Vector Machines on ARM® Cortex®-M4 microcontrollers.

CHAPTER 5

IoT Sensor Applications

New IoT-based products are being introduced every day [122, 123, 124, 125]. In this chapter, we introduce cloud-based platforms and discuss emerging applications.

5.1 CLOUD PLATFORMS

Cloud-based solutions offer the advantages of easy scalability and universal access. Storage, # of CPUs and communications throughput can all be scaled with a few clicks of a button. Machine learning capabilities are offered as standard services. IBM, Google, Microsoft, and Amazon are four companies often mentioned relative to the sensor connectivity and AI spaces.

Leaf-node devices are often resource constrained with limited storage capability. Data is transferred to the cloud using protocols such as Message Queuing Telemetry Transport (MQTT) and Advanced Message Queuing Protocol (AMQP). Figure 5.1 shows the network architecture for a Microsoft Azure IoT application [120]. The central IoT hub consolidates data from multiple sensor devices. These can be connected either directly via AMQP or indirectly via protocol and field gateways.

In this example, a FRDM-K64F development board collects and processes sensor data. Results are then transmitted via an on-board Ethernet port running the AMQP protocol directly to the Azure IoT Hub. Sensor devices utilizing other Ethernet protocols can be connected via Azure IoT protocol gateways, which handle the translation process from one protocol to another. MQTT [126] and AMQP [127] are light-weight internet protocols. They are binary protocols, and the payloads are more compact than HTTP.

Field gateways are useful for the devices that use serial communication protocols such as Bluetooth, USB, I^2C, and SPI. These require some type of hardware/software bridge for connection to the internet. Standard PCs, tablets, and phones may be used for this purpose.

Figure 5.2 illustrates the IBM Watson IoT platform with sensor devices. There two classes of devices: managed and unmanaged. Managed devices can perform operations such as firmware download and updates. Unmanaged devices are limited to sending and receiving data, commands, and events. Once the data is in the cloud, Bluemix IoT services provide a variety of data analytics options.

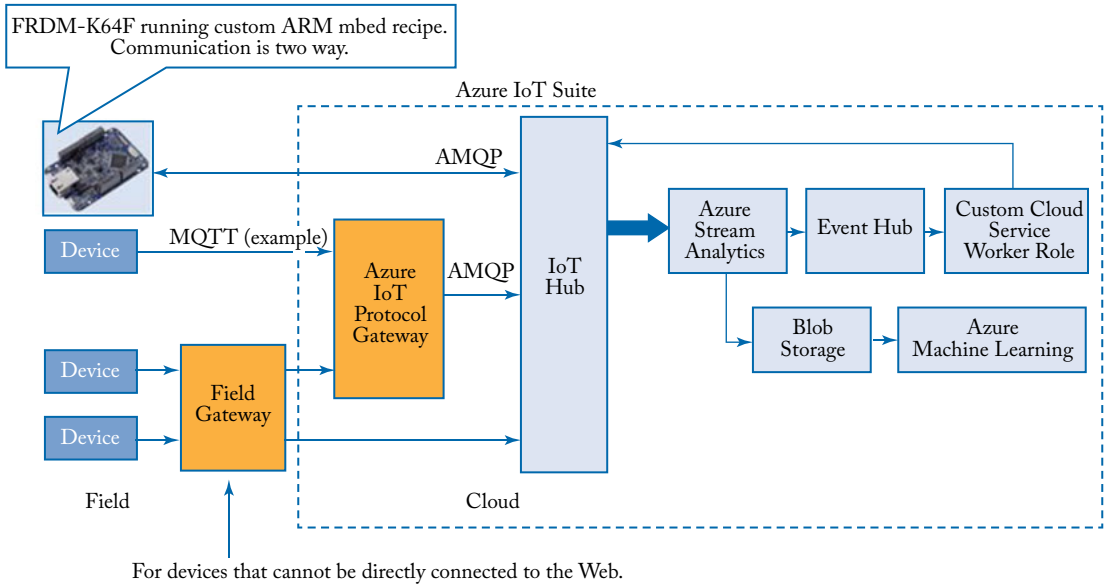

Figure 5.1: Microsoft azure architecture (Adapted from `https://azure.microsoft.com/en-us/`).

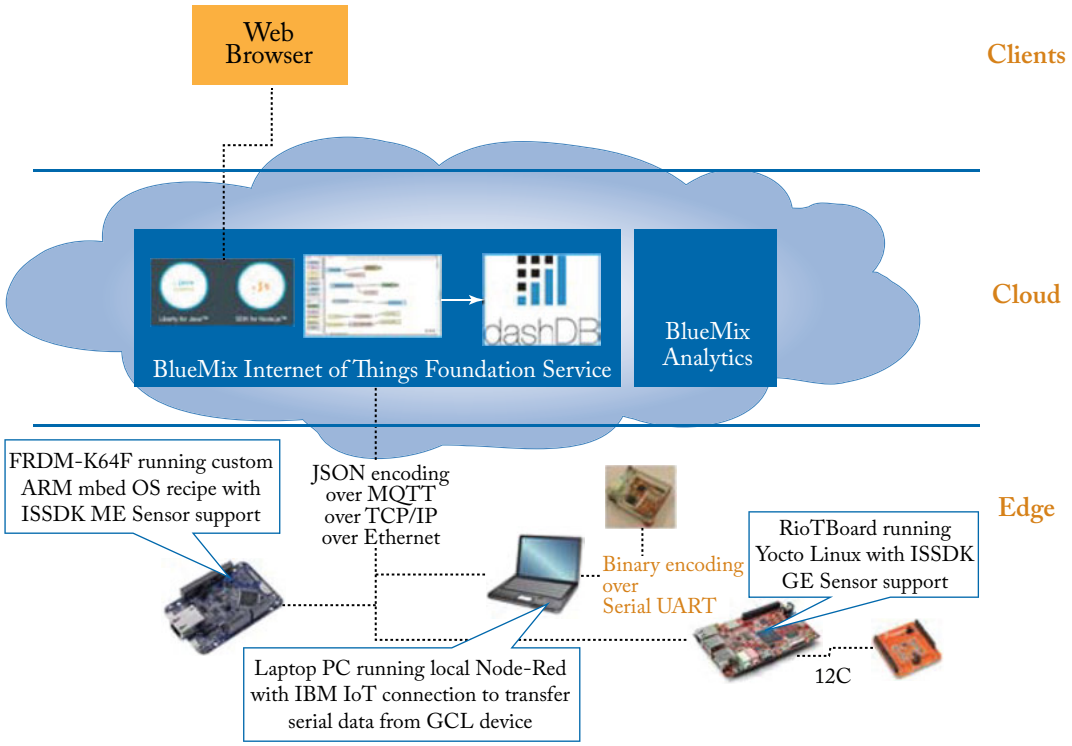

Figure 5.2: IBM Watson IoT platform: data analytics in the cloud (Adapted from the IBM Bluemix website).

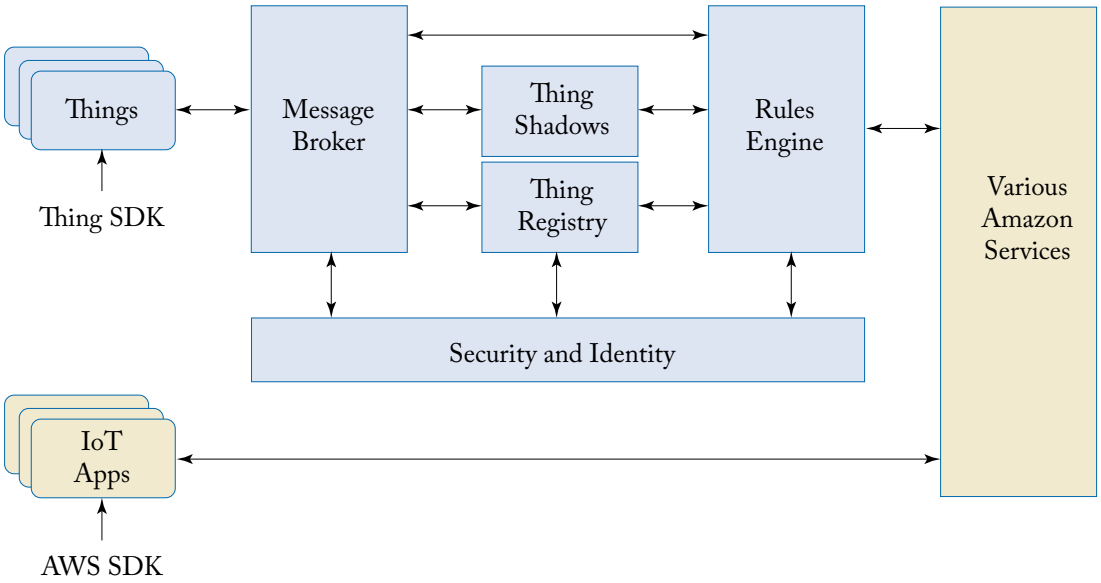

Figure 5.3: Amazon web services IoT (Adapted from `http://docs.aws.amazon.com/iot/latest/developerguide/aws-iot-how-it-works.html`).

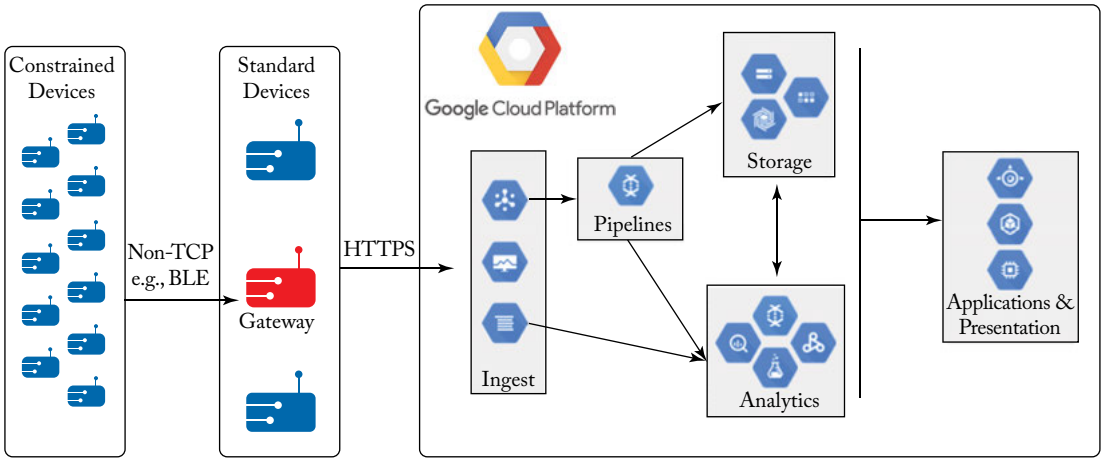

Figure 5.4: Google Cloud Platform (Sourced from `https://cloud.google.com/solutions/architecture/streamprocessing` under Creative Commons Attribution 3.0 License).

Table 5.1: Cloud-based IoT platform comparisons

Feature	Microsoft Azure	IBM Bluemix, SPSS Modeler	Amazon AWS	Google Cloud
Links	https://studio.azureml.net/	console.ng.bluemix.net www.ibm.com/analytics	http://aws.amazon.com/iot/	www.tensorflow.org https://cloud.google.com/ml/
Protocol	AMQP MQTT via protocol Gateway	MQTT	MQTT HTTP RESTful	REST APIs, GRPC
Security	Shared keys Transport Layer Security	Token-based authentication	X509 Certificates Transport Layer Security	SSL/TLS and time limited authentication token
Devices	Many via Microsoft Azure Certified Internet of Things partners	mbed/NXP K64F Arduino Raspberry Pi TI Launchpad FireFly	Raspberry Pi 2 Model B Arduino Yun Linux mbed	None listed
Programming	R, Python inside ML. Java, Node.js, PHP, Python, Ruby for web	SPSS Modeler supports R. Compatible BlueMix runtimes include Node.js, Liberty for Java, Go, PHP, Python, Ruby on Rails, Ruby Sinatra	SDK and CLI-based. SDKs for Java, .NET, Python, PHP, JavaScript & Ruby	TensorFlow, callable from C, Python, … Go, Java, Lua, JavaScript, R. The TensorBoard tool can build graphs of TensorFlow systems after the fact
Model Export/ Import	N/A	PMML. Streams objects can be uploaded to BlueMix	N/A	*Some* PMML preprocessing supported
Model Types	SVM, PCA, decision tree, logistic regression, neural nets, perceptron, Bayes binary classification, K-means, linear, ordinal and Poisson regressions, etc.	Random trees, decision list, regression, PCA, neural net, discriminants, logistic regression, GLMM, GLE, SVM, GLE, Bayes Network, K-means, etc.	Logistic Regression Linear Regression	D.I.Y. using TensorFlow
Environment	Completely cloud-based	SPSS Modeler available for local machine or cloud (free trial does not support cloud-based)	Amazon ML console, CLI and API. AWS SDKs	Textual, although Google does provide the TensorBoard tool for visualizing graphs

Other example platforms include Amazon Web Services IoT and Google Cloud Platform as shown in Fig. 5.3 and Fig. 5.4, respectively. Sensor devices, either managed or unmanaged, are connected through gateways to push sensor data. Communication protocols are based on TCP/IP, MQTT, AMQP, and JSON. Analytics and IoT services are performed in the cloud. Table 5.1 compares the different cloud-based IoT platforms.

5.2 AUTOMOTIVE INDUSTRY

Automotive semiconductor manufacturers describe their products as belonging to various standard subsystems, including

- ABS (Anti-lock Braking Systems),
- ADAS (Advanced Driver Assistance System),
- ESC (Electronic Stability Control), and
- V2X (Vehicle-to-Everything) Communications.

These are dependent upon a large assortment of sensor devices, including accelerometers, magnetometers, gyroscopes, GPS, pressure, light, temperature sensors, vision, RADAR, and LIDAR. These sensors generate continuous streams of data, which must be processed and consumed in real-time (Fig. 5.5).

In addition to the subsystems listed above, data analytics and machine learning techniques operating on these data streams provide 360° situational awareness. This includes object detection and tracking and localization of the vehicle within a map of its surroundings.

As the trend towards autonomous cars continues, there will be many opportunities in R&D, along with new business models for the connected car and IoT automotive industry [128]. In the past, automotive sensors were only used to service an *individual* vehicle, supporting standard safety, infotainment, and control functions. As we move forward, connected vehicles will communicate using V2V (Vehicle-to-Vehicle—Fig. 5.6) and V2I (Vehicle-to-Infrastructure) technologies. Collectively, V2V + V2I are referred to as V2X (Vehicle-to-Everything).

Dedicated Short-Range Communications (DSRC) is a short to medium range system providing very high data rates for vehicle safety applications. The Federal Communication Commission has allocated 75 MHz of spectrum bandwidth at 5.9 GHz radio frequency band for use by DSRC. The U.S. Department of Transportation provides supporting resources and references online at the Intelligent Transportation Systems Joint Program Office website [129]. Potential applications include: emergency warning system for vehicles, cooperative adaptive cruise control, cooperative collision warning, electronic parking payments, and so on.

Work on autonomous vehicles is moving quickly, and there are already several prototype autonomous vehicles in active development. These include Google's Self Driving Car [130], Tesla Autopilot [131], Uber [132], and other automotive makers.

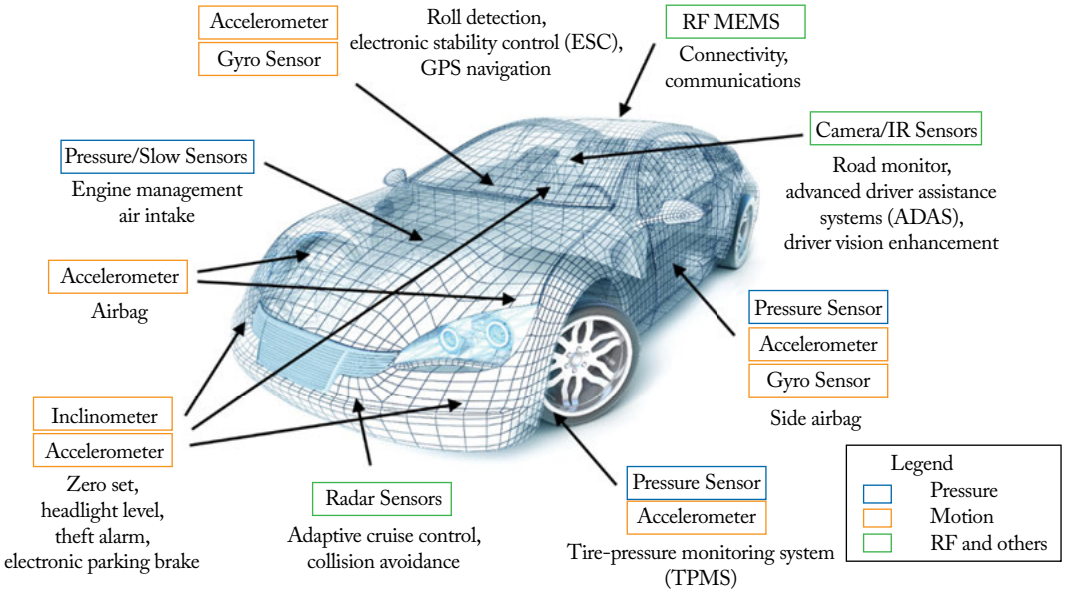

Figure 5.5: MEMS sensors in automotive.

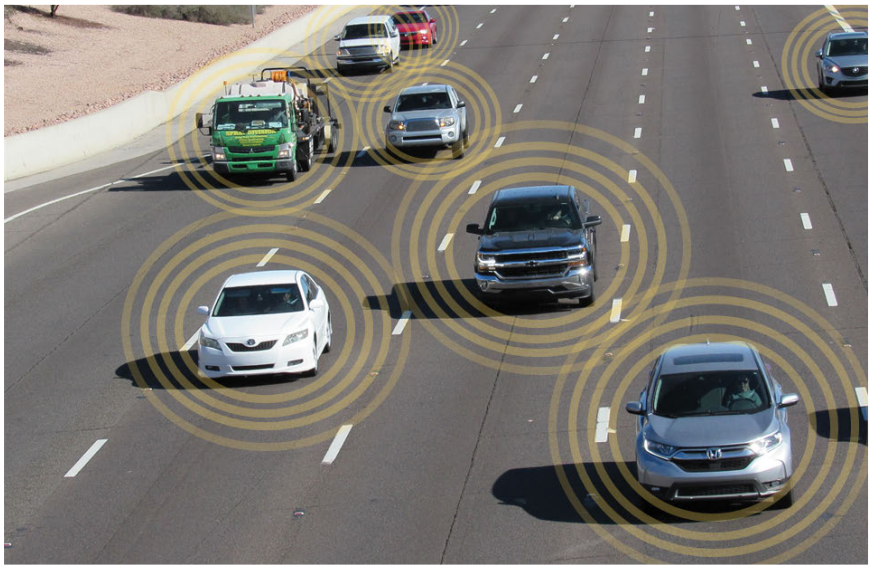

Figure 5.6: Connected vehicles.

5.3 UNMANNED AERIAL VEHICLES (UAV)

Also known as Unmanned Aerial Systems (UAS), or more commonly "drones," these systems share many of the technologies used in the automotive market. Whether talking about small toys you can hold in your hand, or high-end systems like that shown in Fig. 5.7, the basic concepts are the same.

Figure 5.7: High-end UAV.

UAVs come in 3, 4, 5, 6, and 8 motor configurations. At the heart of the system is an inertial sensor fusion system like those discussed in Section 3.5.2. Attitude, altitude, speed, and rotation can all be managed by controlling the relative speed of the UAV motors. Consider the case of a quadcopter with four unidirectional motors, which might be schematically shown as Fig. 5.8.

Notice that motors M3 and M4 rotate in a clockwise direction. Motors M1 and M2 rotate in a counter-clockwise rotation. Upward thrust is proportional to the total thrust of all motors:

$$T \sim \sum_{n=1}^{4} bT_n, \tag{5.1}$$

where b is a design dependent proportionality constant. Angular torque about the X axis can be modeled as

$$\tau_x \sim l(T_1 + T_3 - T_2 - T_4) \tag{5.2}$$

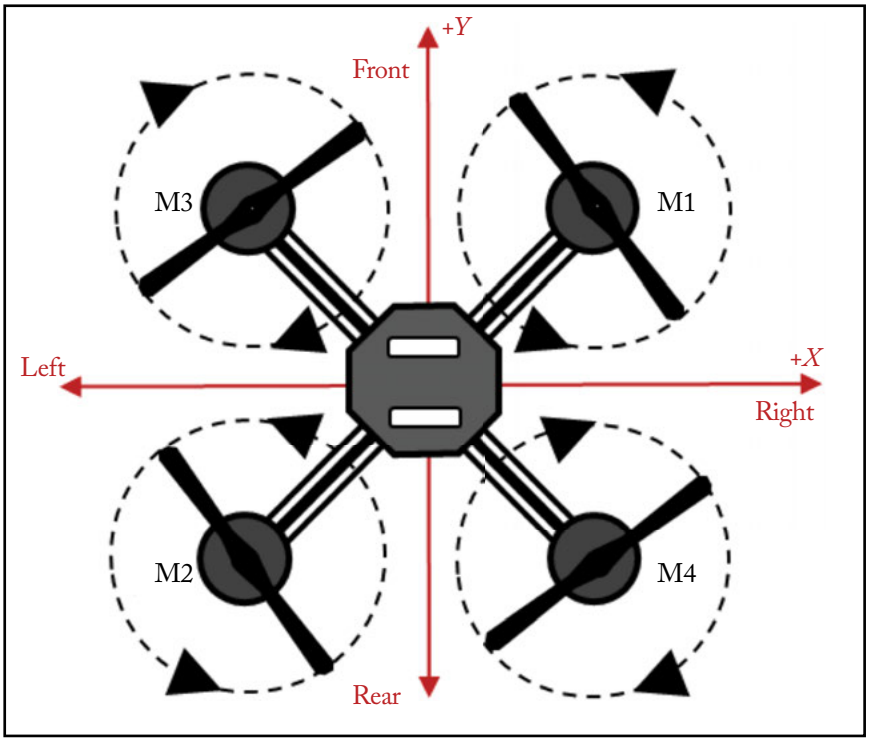

Figure 5.8: Quadcopter motor arrangement.

and angular torque about the Y axis as

$$\tau_y \sim l(T_2 + T_3 - T_1 - T_4), \tag{5.3}$$

where l is a design dependent proportionality constant.

Angular torque about the Z axis results from propeller drag, and uses a different proportionality constant k.

$$\tau_z \sim k(T_3 + T_4 - T_1 - T_2). \tag{5.4}$$

Collectively, these expressions can be modelled in matrix form:

$$\begin{bmatrix} b & b & b & b \\ l & -l & l & -l \\ -l & l & l & -l \\ -k & -k & k & k \end{bmatrix}^{-1} \begin{bmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix}. \tag{5.5}$$

Once the UAV attitude (orientation) is known, and understanding the basic physics of (5.5), it is possible to write a control loop in quaternion form to manage motor drivers to force the quadcopter to an orientation specified via RC control from the operator [133].

The inertial sensor fusion system of Section 3.5.2 also yields estimates for device acceleration. Merging that with altitude estimates provided by pressure sensors/altimeters (Section 2.4) and location estimates from GPS, and we have the equipment for a full computerized navigation system. Additional technologies to aide in this effort include vision, LIDAR (Light Detection and Ranging) and RADAR (RAdio Detection and Ranging). UAVs using these technologies can now be used for Simultaneous Localization and Mapping (SLAM [134]).

Advanced UAVs are finding their way into disaster recovery situations, farm management (monitoring soil, irrigation and livestock), pipeline monitoring, aerial imaging, and more.

5.4 MANUFACTURING AND PROCESSING INDUSTRY

Machine Condition Monitoring (MCM) and Condition-Based Maintenance (CBM) are two terms used to describe the use of sensors and advanced data analysis to monitor manufacturing processes for early fault detection, diagnosis, and prediction [1, 2, 25, 26, 61]. The goal is to schedule repairs before key machinery fails and brings production to a halt.

To do this, massive amount of sensor data are collected, meaningful features and patterns are extracted, and then predictive analytics are performed. Condition-based maintenance with sensor data analytics can reduce maintenance cost significantly because early symptoms of mechanical failures can be detected before they affect production. Machine learning algorithms can be integrated locally or via cloud-based platforms

5.5 HEALTHCARE AND WEARABLES

Key marketplaces for sensors and advanced diagnostics are in healthcare and wearables. Advanced ML-based techniques and low-cost sensors are transforming the way people obtain diagnostic information about their own body processes. Instead of going to a laboratory setting, they will be able to perform key tests right in the home, transferring data to their doctors electronically when needed for review.

Major advantages of using sensors are in prevention, easy access of disease diagnostics, and prediction. Sensors can measure activities of patients or athletes [7, 10, 11]. They can analyze specific patterns and provide predictive monitoring.

5.6 SMART CITY AND ENERGY

Embedded sensor technology enables sensors to be placed almost everywhere in urban environments, including buildings, lights, parking lots, roads, and even electricity meter networks. Collecting sensor data continuously in real time allows cities to optimize resource allocation and maintenance [135, 136, 137, 138, 139, 140, 141]. Smart lighting, electricity power distribution, water consumption, transportation, and traffic load management are examples of features that can be achieved via sensor data analytics and IoT connectivity. These features are also related to city-wide energy efficiency.

CHAPTER 6

Concluding Remarks and Summary

This book focused on sensor analysis for IoT applications. In Chapter 2, we introduced basic operational principles of select types of sensors, namely, accelerometer, magnetometer, gyroscope, and pressure sensors because those sensors are heavily used in IoT applications. We also addressed limitations and potential solutions that one may encounter when using these sensors in practical IoT devices. We described where sensors should be located and how that location affects results. We discussed magnetic interference, and how (in some cases) that can be compensated for mathematically. In Chapter 3, we described sensor fusion algorithms. It is often insufficient to fully describe a problem with a single sensor, as individual sensor types have their own limitations. With orientation estimation, Kalman filtering, and virtual gyroscope examples, we explained how sensor fusion can overcome the limitations and improve sensor operations. We also briefly described the mathematical background used for the sensor fusion algorithms. MATLAB scripts and software tools were also introduced to help the readers develop their own sensor fusion applications. In Chapter 4, we reviewed machine learning algorithms to help the readers understand fundamentals of analytics for sensor data. Example software tools also can help the readers practice machine learning algorithms and sensor data analytics. In Chapter 5, we described several domains of IoT applications with relevant software tools.

There are hundreds, if not thousands, of sensor types available for use in an almost infinite set of applications. The reader is encouraged to delve further into the bibliography to explore these topics in depth. We provide additional references to help the reader gain more understanding in signal processing and machine learning for image sensor data in [148, 149, 150, 151]. Additional references with speech signal are given in [93, 94, 152, 153, 154, 169].

Bibliography

[1] A. K. S. Jardine, D. Lin, and D. Banjevic, A review on machinery diagnostics and prognostics implementing condition-based maintenance, *Mechanical Systems and Signal Processing*, vol. 20, no. 7, pp. 1483–1510, 2006. DOI: 10.1016/j.ymssp.2005.09.012. 2, 11, 68, 94

[2] C. Aldrich and L. Auret, *Unsupervised Process Monitoring and Fault Diagnosis with Machine Learning Methods*, Springer, 2013. DOI: 10.1007/978-1-4471-5185-2. 2, 11, 94

[3] X. Long, B. Yin, and R. M. Aarts, Single-accelerometer-based daily physical activity classification, in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2009. DOI: 10.1109/iembs.2009.5334925. 2, 11

[4] X. Yun and E. R. Bachmann, Design, implementation, and experimental results of a quaternion-based Kalman filter for human body motion tracking, *IEEE Transactions on Robotics*, vol. 22, no. 6, pp. 1216–1227, 2006. DOI: 10.1109/tro.2006.886270. 2, 3, 59

[5] A. Avci, S. Bosch, M. MArin-Perianu, R. Marin-Perianu, and P. Havinga, Activity recognition using inertial sensing for healthcare, wellbeing, and sports applications: A survey, in *23th International Conference on Architecture of Computing Systems*, 2010. 2

[6] M. Zhang and A. A. Sawchuk, Human daily activity recognition with sparse representation using wearable sensors, *IEEE Journal of Biomedical and Health Informatics*, vol. 17, no. 3, pp. 553–560, 2013. DOI: 10.1109/jbhi.2013.2253613. 2

[7] H. Song, J. Thiagarajan, K. N. Ramamurthy, A. Spanias, and P. Turaga, Consensus inference on mobile phone sensors for activity recognition, in *IEEE ICASSP*, 2016. DOI: 10.1109/icassp.2016.7472086. 2, 94

[8] I. Anderson, J. Maitland, S. Sherwood, L. Barkhuus, M. Chalmers, M. Hall, B. Brown, and H. Muller, Shakra: Tracking and sharing daily activity levels with unaugmented mobile phones, *Mobile Networks and Applications*, vol. 12, no. 2–3, pp. 185–199, 2007. DOI: 10.1007/s11036-007-0011-7. 2

[9] C. Orwat, A. Graefe, and T. Faulwasser, Towards pervasive computing in healthcare—A literature review, *BMC Medical Informatics and Decision Making*, vols. 1–18, no. 26, p. 26, 2008. DOI: 10.1186/1472-6947-8-26. 2

[10] S. Patel, H. Park, P. Bonato, L. Chan, and M. Rodgers, A review of wearable sensors and systems with application in rehabilitation, *Journal of Neuroengineering and Rehabilitation*, vol. 9, no. 21, pp. 1–17, 2012. DOI: 10.1186/1743-0003-9-21. 2, 94

[11] D. Rajan, A. Spanias, S. Ranganath, M. Banavar, and P. Spanias, Health monitoring laboratories by interfacing physiological sensors to mobile android devices, in *IEEE FIE*, 2013. DOI: 10.1109/fie.2013.6684987. 2, 94

[12] J. P. Lynch, A summary review of wireless sensors and sensor networks for structural health monitoring, *The Shock and Vibration Digest*, vol. 38, no. 2, pp. 91–128, 2006. 2

[13] J. Yick, B. Mukherjee, and D. Ghosal, Wireless sensor network survey, *Computer Networks*, vol. 52, no. 12, pp. 2292–2330, 2008. DOI: 10.1016/j.comnet.2008.04.002. 2

[14] A. Kim and M. F. Golnaraghi, A quaternion-based orientation estimation algorithm using an inertial measurement unit, in *PLANS, Position Location and Navigation Symposium*, 2004. DOI: 10.1109/plans.2004.1309003. 3, 59

[15] D. Gebre-Egziabher, R. C. Hayward, and J. D. Powell, Design of multi-sensor attitude determination systems, *IEEE Transactions on Aerospace and Electronic Systems*, vol. 40, no. 2, pp. 627–649, 2004. DOI: 10.1109/taes.2004.1310010. 3, 59

[16] NXP Sensor Fusion Library. `http://www.nxp.com/products/sensors/nxp-sensor-fusion:XTRSICSNSTLBOX` 3, 59

[17] Android Sensor Fusion Library. `https://android.googlesource.com/platform/frameworks/native/+/b398927/services/sensorservice/` 3, 59

[18] J. Favre, B. M. Jolles, O. Siegrist, and K. Aminian, Quaternion-based fusion of gyroscopes and accelerometers to improve 3D angle measurement, *Electronics Letters*, vol. 42, no. 11, pp. 612–614, 2006. DOI: 10.1049/el:20060124. 3, 59

[19] X. Yun, E. R. Bachmann, and R. B. McGhee, A simplified quaternion-based algorithm for orientation estimation from earth gravity and magnetic field measurements, *IEEE Transactions on Instrumentation and Measurement*, vol. 57, no. 3, pp. 638–650, 2008. DOI: 10.1109/tim.2007.911646. 3, 59

[20] J. L. Marins, X. Yun, E. R. Bachmann, R. B. McGhee, and M. J. Zyda, An extended Kalman filter for quaternion-based orientation estimation using MARG sensors, in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001. DOI: 10.1109/iros.2001.976367. 3, 59

[21] R. Kannan, Orientation estimation based on LKF using differential state equation, *IEEE Sensors Journal*, vol. 15, no. 11, pp. 6156–6163, 2015. DOI: 10.1109/jsen.2015.2455496. 3, 59

[22] R. G. Valenti, I. Dryanovski, and J. Xiao, Keeping a good attitude: A quaternion-based orientation filter for IMUs and MARGs, *Sensors*, vol. 15, no. 8, pp. 19302–19330, 2015. DOI: 10.3390/s150819302. 3, 59

[23] M. Pedley and M. Stanley, NXP Application Note AN5023: Sensor Fusion Kalman Filters, NXP Semiconductor N. V., 2015. 3, 57, 59

[24] J. Gantz and D. Reinsel, The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east, *IDC iView: IDC Anal. Future*, 2012. 4

[25] C. C. Aggarwal, *Managing and Mining Sensor Data*, Springer, 2013. DOI: 10.1007/978-1-4614-6309-2. 4, 94

[26] J. Lee, M. Stanley, A. Spanias, and C. Tepedelenlioglu, Integrating machine learning in embedded sensor systems for internet-of-things applications, in *IEEE International Symposium on Signal Processing and Information Technology*, Limassol, Cyprus, 2016. DOI: 10.1109/isspit.2016.7886051. 4, 80, 94

[27] B. Scholkopf and A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*, MIT Press, 2001. 4, 76

[28] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning with Applications in R*, Springer, 2014. 4, 76

[29] X. Zhang, X. Chen, W.-H. Wang, J.-H. Yang, V. Lantz, and K.-Q. Wang, Hand gesture recognition and virtual game control based on 3D accelerometer and EMG sensors, in *14th International Conference on Intelligent User Interfaces*, 2009. DOI: 10.1145/1502650.1502708. 11

[30] R. C. Luo and M. G. Kay, A tutorial on multisensor integration and fusion, in *IEEE 16th Annual Conference of Industrial Electronics Society (IECON)*, 1990. DOI: 10.1109/iecon.1990.149228. 29

[31] A. K. Bourke, J. V. O'Brien, and G. M. Lyons, Evaluation of a threshold-based tri-axial accelerometer fall detection algorithm, *Gait and Posture*, vol. 26, pp. 194–199, 2007. DOI: 10.1016/j.gaitpost.2006.09.012. 11

[32] T. J. Davis, N. C. Macdonald, and G. J. Galvin, Microelectromechanical accelerometer for automotive applications, United States Patent US6199874 B1, March 13, 2001. 11

[33] A. V. Rao, *Dynamics of Particles and Rigid Bodies: A Systematic Approach*, Cambridge University Press, 2005. DOI: 10.1017/cbo9780511805455. 12, 13

[34] NOAA National Centers for Environmental Information, The World Magnetic Model, National Oceanic and Atmospheric Administration. `https://www.ngdc.noaa.gov/geomag/WMM/` 13

[35] D. Gebre-Egziabher, Magnetometer autocalibration leveraging measurement locus constraints, *Journal of Aircraft*, vol. 44, no. 4, pp. 1361–1368, 2007. DOI: 10.2514/1.27118. 17

[36] J. F. Vasconcelos, G. Elkaim, C. Silvestre, P. Oliveira, and B. Cardeira, A geometric approach to strapdown magnetometer calibration in sensor frame, *IEEE Transactions on Aerospace and Electronic Systems,* vol. 47, no. 2, pp. 1293–1306, 2011. DOI: 10.1109/taes.2011.5751259. 17

[37] D. Gebre-Egziabher, G. H. Elkaim, J. D. Powell, and B. W. Parkinson, A nonlinear two-step estimation algorithm for calibrating solid-state strapdown magnetometers, in *International Conference on Integrated Navigation Systems (ICINS)*, pp. 290–297, 2001. 17

[38] E. Dorveaux, D. Vissiere, A. P. Martin, and N. Petit, Iterative calibration method for inertial and magnetic sensors, in *48th IEEE Conference on Decision and Control (CDC)*, held jointly with *28th Chinese Control Conference*, pp. 8296–8303, Shanghai, 2009. DOI: 10.1109/cdc.2009.5399503. 17

[39] M. Stanley, *Physical Design Considerations for Product Incorporating Inertial and Magnetic Sensors*, San Jose, Presentation at Design West, 2012. 19

[40] J. Fraden, *Handbook of Modern Sensors*, Springer, 2016. DOI: 10.1007/978-1-4419-6466-3. 23

[41] R. C. Luo and M. G. Kay, A tutorial on multisensor integration and fusion, in *16th Annual Conference of IEEE Industrial Electronics Society (IECON)*, 1990. DOI: 10.1109/iecon.1990.149228. 29

[42] D. L. Hall and J. Llinas, An intorduction to multisensor data fusion, *Proc. of the IEEE*, vol. 85, no. 1, pp. 6–23, 1997. DOI: 10.1109/5.554205. 29

[43] J. D. Irwin, *The Industrial Electronics Handbook*, CRC Press, 1997. 29

[44] E. Bergamini, G. Ligorio, A. Summa, G. Vannozzi, A. Cappozzo, and A. M. Sabatini, Estimating orientation using magnetic and inertial sensors and different sensor fusion approaches: Accuracy assessment in manual and locomotion tasks, *Sensors*, vol. 14, no. 10, pp. 18625–49, 2014. DOI: 10.3390/s141018625. 30

[45] S. Ounpuu, R. B. Davis, and P. A. DeLuca, Joint kinetics: Methods, interpretation and treatment decision-making in children with cerebral palsy and myelomeningocele, *Gait and Posture*, vol. 4, no. 1, pp. 62–78, 1996. DOI: 10.1016/0966-6362(95)01044-0. 30

[46] K. Aminian, C. Trevisan, B. Najafi, H. Dejnabadi, C. Frigo, E. Pavan, A. Telonio, F. Cerati, E. Marinoni, P. Robert, and P.-F. Leyvraz, Evaluation of an ambulatory system

for gait analysis in hip osteoarthritis and after total hip replacement, *Gait and Posture*, vol. 20, no. 1, pp. 102–107, 2004. DOI: 10.1016/s0966-6362(03)00093-6. 30

[47] M. A. Mont, T. M. Seyler, P. S. Ragland, R. Starr, J. Erhart, and A. Bhave, Gait analysis of patients with resurfacing hip arthroplasty compared with hip osteoarthritis and standard total hip arthroplasty, *The Journal of Arthroplasty*, vol. 22, no. 1, pp. 100–108, 2007. DOI: 10.1016/j.arth.2006.03.010. 30

[48] J. Howcroft, J. Kofman, and E. D. Lemaire, Review of fall risk assessment in geriatric populations using inertial sensors, *Journal of Neuroengineering and Rehabilitation*, vol. 10, no. 91, 2013. DOI: 10.1186/1743-0003-10-91. 30

[49] H. J. Luinge and P. H. Veltink, Inclination measurement of human movement using a 3D accelerometer with autocalibration, *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 12, no. 1, pp. 112–121, 2004. DOI: 10.1109/tnsre.2003.822759. 30

[50] H. Zhou and H. Hu, Human motion tracking for rehabilitation—A survey, *Biomedical Signal Processing and Control*, vol. 3, no. 1, pp. 1–18, 2008. DOI: 10.1016/j.bspc.2007.09.001. 30

[51] B. Barshan and H. F. Durrant-Whyte, Inertial navigation systems for mobile robots, *IEEE Transactions on Robotics and Automation,* vol. 11, no. 3, pp. 328–342, 1995. DOI: 10.1109/70.388775. 30

[52] L. Ojeda and J. Borenstein, Flexnav: Fuzzy logic expert rule-based position estimation for mobile robots on rugged terrain, in *IEEE International Conference on Robotics and Automation (ICRA)*, 2002. DOI: 10.1109/robot.2002.1013380. 30

[53] S. K. Hong, Fuzzy logic based closed-loop strapdown attitude system for unmanned aerial vehicle, *Sensors and Actuators A Physical*, vol. 107, no. 2, pp. 109–118, 2003. DOI: 10.1016/s0924-4247(03)00353-4. 30

[54] D. Titterton and J. L. Weston, Strapdown inertial navigation technology, *The Institution of Electrical Engineers (IET)*, 2004. DOI: 10.1049/pbra017e. 30

[55] S. Beauregard, Omnidirectional pedestrian navigation for first responders, in *4th Workshop on Positioning, Navigation and Communication (WPNC)*, 2007. DOI: 10.1109/wpnc.2007.353609. 30

[56] E. A. Heinz, K. S. Kunze, M. Gruber, D. Bannach, and P. Lukowicz, Using wearable sensors for real-time recognition tasks in games of martial arts—An initial experiment, in *IEEE Symposium on Computational Intelligence and Games*, 2006. DOI: 10.1109/cig.2006.311687. 30, 45

[57] J. B. Kuipers, *Quaternions and Rotation Sequences*, Princeton, NJ, Princeton University Press, 1999. DOI: 10.7546/giq-1-2000-127-143. 37, 38, 39, 45, 51

[58] T. Ozyagcilar, Implementing a tilt-compensated eCompass using accelerometer and magnetometer sensors, *NXP Application Note, AN4248*, 2015. 54

[59] G. Wahba, A least squares estimate of satellite attitude, *Society for Industrial and Applied Mathematics (SIAM) Review*, vol. 7, no. 3, pp. 409–409, 1965. DOI: 10.1137/1007077. 54, 61

[60] M. Pedley and M. Stanley, NXP Application Note AN5018 Basic Kalman Filter Theory, NXP Semiconductor N. V., 2015. 59

[61] C. C. Aggarwal, *Outlier Analysis*, Springer, NY, 2013. DOI: 10.1007/978-1-4614-6396-2. 67, 94

[62] R. J. A. Little and D. B. Rubin, *Statistical Analysis with Missing Data*, New York, NY, John Wiley & Sons, Inc., 1986. DOI: 10.1002/9781119013563. 67

[63] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*, Englewood Cliffs, NJ, Prentice Hall, 1985. 67

[64] S. O. Haykin, *Adaptive Filter Theory*, Prentice Hall, 2001. 67

[65] A. H. Sayed, *Fundamentals of Adaptive Filtering*, Wiley, 2003. 67

[66] A. Spanias, *Digital Signal Processing: An Interactive Approach*, 2nd ed., Morrisville, NC, Lulu Press, 2014. 67

[67] L. Cohen, G. Avrahami-Bakish, M. Last, A. Kandel, and O. Kipersztok, Real-time data mining of non-stationary data streams from sensor networks, *Information Fusion*, vol. 9, no. 3, pp. 344–353, 2008. DOI: 10.1016/j.inffus.2005.05.005. 67

[68] C. O'Reilly, A. Gluhak, M. A. Imran, and S. Rajasegarar, Anomaly detection in wireless sensor networks in a non-stationary environment, *IEEE Communications Surveys and Tutorials*, vol. 16, no. 3, pp. 1413–1432, 2014. DOI: 10.1109/surv.2013.112813.00168. 67

[69] J. R. Blough, A survey of DSP methods for rotating machinery analysis, what is needed, what is available, *Journal of Sound and Vibration*, vol. 262, no. 3, pp. 707–720, 2003. DOI: 10.1016/s0022-460x(03)00118-4. 68

[70] R. B. Randall, *Vibration-based Condition Monitoring*, John Wiley & Sons, Ltd., 2010. DOI: 10.1002/9780470977668. 68

[71] A. Heng, S. Zhang, A. C. C. Tan, and J. Mathew, Rotating machinery prognostics: State of the art, challenges and opportunities, *Mechanical Systems and Signal Processing*, vol. 23, no. 3, pp. 724–739, 2009. DOI: 10.1016/j.ymssp.2008.06.009. 68

[72] I. El-Thalji and E. Jantunen, A summary of fault modelling and predictive health monitoring of rolling element bearings, *Mechanical Systems and Signal Processing*, vols. 60–61, pp. 252–272, 2015. DOI: 10.1016/j.ymssp.2015.02.008. 68

[73] C. L. Nikias and A. P. Petropulu, *Higher-order Spectral Analysis: A Nonlinear Processing Framework*, Prentice Hall, 1993. 68

[74] C. L. Nikias and J. M. Mendel, Signal processing with higher-order spectra, *IEEE Signal Processing Magazine*, vol. 10, no. 3, pp. 10–37, 1993. DOI: 10.1109/79.221324. 68

[75] P. Stoica and R. L. Moses, *Spectral Analysis of Signals*, Pearson, Prentice Hall, 2005. 68, 69

[76] J. M. Mendel, Tutorial on higher-order statistics (spectra) in signal processing and system theory: Theoretical results and some applications, *Proc. of the IEEE*, vol. 79, no. 3, pp. 278–305, 1991. DOI: 10.1109/5.75086. 68

[77] I. Daubechies, *Ten Lectures on Wavelets*, Philadelphia, PA, Society for Industrial and Applied Mathematics, 1992. DOI: 10.1137/1.9781611970104. 69

[78] Z. K. Peng and F. L. Chu, Application of the wavelet transform in machine condition monitoring and fault diagnostics: A review with bibliography, *Mechanical Systems and Signal Processing*, vol. 18, no. 2, pp. 199–221, 2004. DOI: 10.1016/s0888-3270(03)00075-x. 69

[79] J. S. Walker, *A Primer on WAVELETS and their Scientific Applications*, 2nd ed., Boca Raton, FL, CRC Press, 2008. DOI: 10.1201/9781584887461. 69

[80] I. T. Jolliffe, *Principal Component Analysis*, Springer Verlag, 1986. DOI: 10.1007/978-1-4757-1904-8. 70

[81] I. Guyon and A. Elisseeff, An introduction to variable and feature selection, *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003. 70

[82] A. Hyvarinen and E. Oja, Independent component analysis: Algorithms and application, *Neural Networks*, vol. 13, no. 4–5, pp. 411–430, 2000. DOI: 10.1016/s0893-6080(00)00026-5. 70

[83] B. Scholkopf, A. J. Smola, and K-R. Muller, Kernel principal component analysis, in *Advances in Kernel Methods*, pp. 327–352, Cambridge, MA, MIT Press, 1999. DOI: 10.1007/bfb0020217. 70

[84] A. V. Oppenheim and R. W. Schafer, From frequency to quefrency: A history of the cepstrum, *IEEE Signal Processing Magazine*, vol. 21, no. 5, pp. 95–106, September 2004. DOI: 10.1109/msp.2004.1328092. 70

[85] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Boston, MA, Academic Publishers, 1989. 71

[86] R. Storn and K. Price, Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces, *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997. DOI: 10.1023/A:1008202821328. 71

[87] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, Optimization by simulated annealing, *Science*, vol. 220, no. 4598, pp. 671–680, 1983. DOI: 10.1126/science.220.4598.671. 71

[88] W. Zhao, A. Bhushan, A. D. Santamaria, M. G. Simon, and C. E. Davis, Machine learning: A crucial tool for sensor design, *Algorithms*, vol. 1, no. 2, pp. 130–152, 2008. DOI: 10.3390/a1020130. 71

[89] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, 2nd ed., Academic Press, 2013. DOI: 10.1016/c2009-0-27872-x. 71, 76

[90] G. Kimeldorf and G. Wahba, Some results on tchebycheffian spline functions, *Journal of Mathematical Analysis and Applicactions*, vol. 33, pp. 82–95, 1971. DOI: 10.1016/0022-247x(71)90184-3. 75

[91] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, 6th ed., Boston, MA, Kluwer Academic Publishers, 1991. DOI: 10.1007/978-1-4615-3626-0. 77

[92] Y. Linde, A. Buzo, and R. M. Gray, An algorithm for vector quantizer design, *IEEE Transactions on Communications*, vols. COM-28, no. 1, pp. 84–95, 1980. DOI: 10.1109/t-com.1980.1094577. 77

[93] A. Spanias, T. Painter, and V. Atti, *Audio Signal Processing and Coding*, Wiley, 2007. DOI: 10.1002/0470041978. 77, 95

[94] A. Spanias, Speech coding: A tutorial review, *Proc. of the IEEE*, vol. 82, no. 10, pp. 1541–1582, 1994. DOI: 10.1109/5.326413. 77, 95

[95] T. Painter and A. Spanias, Perceptual coding of digital audio, *Proc. of the IEEE*, vol. 88, no. 4, pp. 451–515, 2000. DOI: 10.1109/5.842996. 77

[96] S. Dudoit and J. Fridlyand, A prediction-based resampling method for estimating the number of clusters in a dataset, *Genome Biology*, vol. 3, no. 7, pp. 0036.1–21, 2002. DOI: 10.1186/gb-2002-3-7-research0036. 79

[97] A. Thalamuthu, I. Mukhopadhyay, X. Zheng, and G. C. Tseng, Evaluation and comparison of gene clustering methods in microarray analysis, *Bioinformatics*, vol. 22, no. 19, pp. 2405–2412, 2006. DOI: 10.1093/bioinformatics/btl406. 79

[98] Y. Bengio, Learning deep architectures for AI, *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009. DOI: 10.1561/2200000006. 79

[99] Y. LeCun, Y. Bengio, and G. Hinton, Deep learning, *Nature,* vol. 521, pp. 436–444, 2015. DOI: 10.1038/nature14539. 79

[100] K. He, X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition, in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. DOI: 10.1109/cvpr.2016.90. 79

[101] J. Schmidhuber, Deep learning in neural networks: An overview, *Neural Networks*, vol. 61, pp. 85–117, 2015. DOI: 10.1016/j.neunet.2014.09.003. 79

[102] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016. 79

[103] A. Krizhevsky, I. Sutskever, and G. E. Hinton, ImageNet classification with deep convolutional neural networks, in *Advances in Neural Information Processing Systems 25*, 2012. DOI: 10.1145/3065386. 79

[104] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, SoK: Towards the Science of Security and Privacy in Machine Learning, November 11, 2016. https://arxiv.org/pdf/1611.03814.pdf 80

[105] A. D. Sarwate and K. Chaudhuri, Signal processing and machine learning with differential privacy, *IEEE Signal Processing Magazine*, pp. 86–94, September 2013. DOI: 10.1109/msp.2013.2259911. 80

[106] Mathworks, Mathworks—Makers of MATLAB and Simulink. https://www.mathworks.com/ 82

[107] Python Software Foundation. https://www.python.org/ 82

[108] R Foundation, The R Project for Statistical Computing. https://www.r-project.org/ 82

[109] T. Galili, What are the Best Machine Learning Packages in R, 2016. https://www.r-bloggers.com/what-are-the-best-machine-learning-packages-in-r/ 82

[110] S. van Buuren, K. Groothuis-Oudshoorn, A. Robitzsch, G. Vink, L. Doove, S. Jolani, R. Schouten, P. Gaffert, and F. Meinfelder, Mice: Multivariate Imputation by Chained Equations in R, 2017. https://cran.r-project.org/web/packages/mice/index.html DOI: 10.18637/jss.v045.i03. 82

[111] T. Therneau, B. Atkinson, and B. Ripley, Rpart: Recursive Partitioning and Regression Trees, 2017. https://cran.r-project.org/web/packages/rpart/index.html 82

[112] T. Hothorn, K. hornik, C. Strobl, and A. Zeileis, Party: A Laboratory for Recursive Partytioning, 2017. https://cran.r-project.org/web/packages/party/index.html 82

[113] M. Kuhn, The Caret Package, 2017. http://topepo.github.io/caret/index.html 82

[114] M. Kuhn, J. Wing, S. Weston, A. Williams, C. Keefer, A. Engelhardt, T. Cooper, Z. Mayer, B. Kenkel, M. Benesty, R. Lescarbeau, A. Ziem, L. Scrucca, Y. Tang, C. Candan, and T. Hunt, Caret: Classification and Regression Training, 2017. https://cran.r-project.org/web/packages/caret/ 82

[115] B. Ripley and W. Venables, Nnet: Feed-forward Neural Networks and Multinomial Log-linear Models, 2016. https://cran.r-project.org/web/packages/nnet/index.html 82

[116] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, G. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, Scikit-learn: Machine Learning in Python, *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. 82

[117] Scikit-learn: Machine Learning in Python. http://scikit-learn.org/stable/index.html 82

[118] Knime Analytics Platform. https://www.knime.com/knime-analytics-platform DOI: 10.15363/thinklab.d33. 82

[119] IBM SPSS Modeler. https://www.ibm.com/us-en/marketplace/spss-modeler 82

[120] Microsoft, Welcome to Azure Machine Learning. https://studio.azureml.net/ 82, 85

[121] TensorFlow—An Open-source Software Library for Machine Intelligence. https://www.tensorflow.org/ 84

[122] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, Internet of Things (IoT): A vision, architectural elements, and future directions, *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013. DOI: 10.1016/j.future.2013.01.010. 1, 85

[123] C. Perera, C. H. Liu, S. Jayawardena, and M. Chen, A survey on Internet of Things from industrial market perspective, *IEEE Access*, vol. 2, pp. 1660–1679, 2014. DOI: 10.1109/access.2015.2389854. 1, 85

[124] C. Perera, C. H. Liu, and S. Jayawardena, The emerging Internet of Things marketplace from an industrial perspective: A survey, *IEEE Transactions on Emerging Topics in Computing*, vol. 3, no. 4, pp. 585–598, 2015. DOI: 10.1109/tetc.2015.2390034. 1, 85

[125] D. X. Li, W. He, and S. Li, Internet of Things in industries: A survey, *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2233–2243, 2014. DOI: 10.1109/tii.2014.2300753. 1, 85

[126] MQTT Version 3.1.1 Plus Errata 01, A. Banks and R. Gupta, Eds., OASIS Standard Incorporating Approved Errata 01, December 10, 2015. 85

[127] OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0, OASIS Standard, 2012. 85

[128] S. Ninan, B. Gangula, M. V. Alten, and B. Sniderman, Who owns the road? The IoT-connected car of today and tomorrow, *Deloitte University Press*, August 18, 2015. 90

[129] United States Department of Transportation, Connected Vehicles. `https://www.its.dot.gov/cv_basics/` 90

[130] Google, Waymo. `https://en.wikipedia.org/wiki/Waymo` 90

[131] Tesla, Autopilot. `https://en.wikipedia.org/wiki/Tesla_Autopilot` 90

[132] Uber, Otto. `https://en.wikipedia.org/wiki/Otto_(company)` 90

[133] E. Fresk and G. Nikolakopoulos, Full quaternion based attitude control for a quadrotor, in *European Control Conference (ECC)*, Zurich, Switzerland, 2013. 93

[134] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, Past, present, and future of simultaneous localization and mapping: Toward the robust perception age, *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016. DOI: 10.1109/tro.2016.2624754. 94

[135] P. Vlacheas, R. Giaffreda, V. Stavroulak, D. Kelaidonis, V. Foteinos, G. Poulios, P. Demestichas, A. Somov, A. R. Biswas, and K. Moessner, Enabling smart cities through a cognitive management framework for the Internet of Things, *IEEE Communications Magazine*, vol. 51, no. 6, pp. 102–111, 2013. DOI: 10.1109/mcom.2013.6525602. 94

[136] S. Rao, D. Ramirez, H. Braun, J. Lee, C. Tepedelenlioglu, E. Kyriakides, D. Srinivasan, J. Frye, S. Koizumi, Y. Morimoto, and A. Spanias, An 18 kW solar array research facility for fault detection experiments, in *IEEE the 18th Mediterranean Electrotechnical Conference (MELECON)*, Lemesos, Cyprus, 2016. DOI: 10.1109/melcon.2016.7495369. 94

[137] S. Peshin, D. Ramirez, J. Lee, H. Braun, C. Tepedelenlioglu, and A. Spanias, A photovoltaic (PV) array monitoring simulator, in *IASTED International Conference on MIC*, 2015. DOI: 10.2316/p.2015.826-029. 94

[138] S. Rao, S. Katoch, P. Turaga, A. Spanias, C. Tepedelenlioglu, R. Ayyanar, H. Braun, J. Lee, U. Shanthamallu, M. Banavar, and D. Srinivasan, A cyber-physical system approach for photovoltaic array monitoring and control, in *IEEE the 8th International Conference on Information Intelligence, Systems and Applications (IEEE IISA)*, Larnaca, 2017. 94

[139] A. Spanias, Solar energy management as an Internet of Things (IoT) Application, in *IEEE the 8th International Conference on Information, Intelligence, Systems and Applications (IISA)*, Larnaca, 2017. 94

[140] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, Internet of Things for smart cities, *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22–32, 2014. DOI: 10.1109/jiot.2014.2306328. 94

[141] P. Bellavista, G. Cardone, A. Corradi, and L. Foschini, Convergence of MANET and WSN in IoT urban scenarios, *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3558–3567, 2013. DOI: 10.1109/jsen.2013.2272099. 94

[142] D. Meeker, Finite Element Method Magnetics (FEMM). `http://www.femm.info/wiki/HomePage` 18

[143] S. Liu and G. Trenkler, Hadamard, Khatri-Rao, Kronecker and other matrix products, *International Journal of Information and Systems Sciences*, vol. 4, no. 1, pp. 160–177, 2008. 53

[144] D. Yu and D. Li, Deep learning and its applications to signal and information processing, *IEEE Signal Processing Magazine*, vol. 28, no. 1, pp. 145–154, 2011. DOI: 10.1109/msp.2010.939038. 79

[145] H. Song, J. J. Thiagarajan, P. Sattigeri, K. N. Ramamurthy, and A. Spanias, A deep learning approach to multiple Kernel fusion, in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017. DOI: 10.1109/icassp.2017.7952565. 79

[146] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups, *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012. DOI: 10.1109/msp.2012.2205597. 18, 79

[147] D. Yu and L. Deng, *Automatic Speech Recognition: A Deep Learning Approach*, Springer, 2014. DOI: 10.1007/978-1-4471-5779-3. 79

[148] J. Thiagarajan, K. Ramamurthy, P. Turaga, and A. Spanias, *Image Understanding Using Sparse Representations*, Mortan & Claypool Publishers, 2014. DOI: 10.2200/s00563ed1v01y201401ivm015. 95

[149] H. Braun, P. Turaga, and A. Spanias, Direct tracking from compressive imagers: A proof of concept, in *IEEE ICASSP*, 2014. DOI: 10.1109/icassp.2014.6855187. 95

[150] J. J. Thiagarajan, K. N. Ramamurthy, P. Sattigery, and A. Spanias, Supervised local sparse coding of sub-image features for image retrieval, in *IEEE ICIP*, 2012. DOI: 10.1109/icip.2012.6467560. 95

[151] P. Sattigeri, J. J. Thiagarajan, K. N. Ramamurthy, and A. Spanias, Implementation of a fast image coding and retrieval system using a GPU, in *IEEE ESPA*, 2012. DOI: 10.1109/espa.2012.6152431. 95

[152] G. Wichern, J. Xue, H. Thornburg, B. Mechtley, and A. Spanias, Segmentation, indexing, and retrieval for environmental and natural sounds, *IEEE Transactions on ASLP*, vol. 18, no. 3, pp. 688–707, 2010. DOI: 10.1109/tasl.2010.2041384. 95

[153] P. Sattigeri, J. J. Thiagarajan, M. Shah, K. N. Ramamurthy, and A. Spanias, A scalable feature learning and tag prediction framework for natural environment sounds, in *48th Asilomar Conference on Signals, Systems and Computers*, 2014. DOI: 10.1109/acssc.2014.7094773. 95

[154] J. Makhoul et al., Vector quantization in speech coding, *Proc. of IEEE*, vol. 73, no. 11, pp. 1551–1588, 1985. DOI: 10.1109/proc.1985.13340. 95

[155] U. Shanthamallu, A. Spanias, C. Tepedelenlioglu, and M. Stanley, A brief survey of machine learning methods and their sensor and IoT applications, *Proc. 8th International Conference on Information, Intelligence, Systems and Applications (IEEE IISA)*, Larnaca, August 2017. 65, 67

[156] A. Spanias, A Brief survey of time and frequency domain filters, part of adaptive signal processing tutorial, *Proc. 7th IISA*, Halkidiki, Greece, July 13–15, 2016. DOI: 10.1109/IISA.2016.7785389. 67, 68

[157] N. Kovvali, M. Banavar, and A. Spanias, *An Introduction to Kalman Filtering with MATLAB Examples, Synthesis Lectures on Signal Processing*, Morgan & Claypool Publishers, Jose Mura, Ed., vol. 6, no. 2, pp. 1–81, September 2013. DOI: 10.2200/s00534ed1v01y201309spr012. 56

[158] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*, Prentice Hall, 1993. 69

[159] A. Skodras, C. Christopoulos, and T. Ebrahimi, The JPEG2000 still image compression standard, *IEEE Signal Processing Magazine*, pp. 36–58, September 2001. DOI: 10.1109/79.952804. 70

[160] A. Akansu and M. J. T. Smith, Eds., *Subband and Wavelet Transforms, Design and Applications*, Kluwer Academic Publishers, Norwell, MA, 1996. DOI: 10.1007/978-1-4613-0483-8. 69

[161] G. Strang and T. Nguyen, *Wavelets and Filter Banks*, Wellesley-Cambridge Press, Wellesley, MA, 1996. 69

[162] N. Ahmed and K. Rao, *Orthogonal Transforms for Digital Signal Processing*, Springer-Verlag, New York, 1975. DOI: 10.1007/978-3-642-45450-9. 70, 77

[163] G. Strang, *Linear Algebra and its Applications*, Academic Press, 1982. 53, 77

[164] D. Roetenberg, H. Luinge, C. Baten, and P. Veltink, Compensation of magnetic disturbances improves inertial and magnetic sensing of human body segment orientation, *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 13, no. 3, pp. 395–505, September 2005. DOI: 10.1109/tnsre.2005.847353. 56

[165] E. Tapia, S. Intille, W. Haskell, K. Larson, J. Wright, A. King, and R. Friedman, Real-time recognition of physical activities and their intensities using wireless accelerometers and a heart rate monitor, *11th IEEE International Symposium on Wearable Computers*, 2007. DOI: 10.1109/iswc.2007.4373774. 11

[166] A. Wixted, D. Thiel, A. Hahn, C. Gore, D. Pyne, and D. James, Measurement of energy expenditure in elite athletes using MEMS-based triaxial accelerometers, *IEEE Sensors Journal*, vol. 7, no. 4, pp. 481–488, April 2007. DOI: 10.1109/jsen.2007.891947. 11

[167] L. Kaelbling, M. Littman, and A. Moore, Reinforcement learning: A survey, *Journal of Artificial Intelligence Research 4*, pp. 237–285, September 1995. 65

[168] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*, CRC Press, Automation and Control Engineering Series, April 2010. DOI: 10.1201/9781439821091. 65

[169] M. Shah, C. Chakrabarti, and A. Spanias, Within and cross-corpus speech emotion recognition using latent topic model-based features, *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 4, 2015. DOI: 10.1186/s13636-014-0049-y. 95

[170] F. Porikli, S. Shan, C. Snoek, R. Sukthankar, and X. Wang, Special issue: Deep learning for visual understanding, *Signal Processing Magazine*, vol. 34, no. 6, Novesmber 2017. DOI: 10.1109/MSP.2017.2744538. 79

[171] L. Deng, Deep learning for AI, plenary talk, *Proc. IEEE ICASSP*, Shangai, March 2016. 79

Authors' Biographies

MICHAEL STANLEY

Michael Stanley received his B.S.E. from Michigan State University in 1980 and his M.S. from Arizona State University in 1986. His career spans over three decades at Motorola Semiconductor, Freescale Semiconductor, and NXP Semiconductor. He is a Senior Member of the IEEE and is listed as inventor or co-inventor on seven patents. Mike was a contributor to the IEEE Standard for Sensor Performance Parameter Definitions (IEEE Std 2700™-2014) as well as the 2nd edition of the *Measurements, Instrumentation, and Sensors Handbook* by CRC Press. He was inducted into the MEMS & Sensors Industry Group Hall of Fame in 2015 and he is an active member of the Industry Advisory Board for the Sensor, Signal, and Information Processing Center (SenSIP) at Arizona State University.

JONGMIN LEE

Jongmin Lee is a Systems & Architecture Engineer at NXP Semiconductor. He received his Ph.D. in Electrical Engineering from Arizona State University in 2017. Previously, he was a graduate research associate at the Sensor, Signal, and Information Processing Center (SenSIP) at Arizona State University. His research interests include signal processing, communications, and machine learning.