

ch3 part 10

The Engineer's Compass: A Study Bible on Measures of Central Tendency in Python

Introduction: Beyond the Average - A Mechatronics Engineer's Guide to Central Tendency

In the world of mechatronics and robotics, success is measured in reliability, precision, and intelligence. A self-driving car must interpret a torrent of data from its cameras, LiDAR, and radar to navigate safely, while a robotic arm on an assembly line must process sensor feedback to perform a delicate task with sub-millimeter accuracy.¹ The foundation of this intelligence is not some arcane magic, but the disciplined application of fundamental statistical principles. The measures of central tendency—mean, median, and mode—are not merely abstract concepts from a textbook; they are the indispensable instruments in an engineer's toolkit. They are the compass used to find the "center," the "truth," or the most typical signal within a sea of noisy, uncertain data.

This study guide is designed to build a robust mental model of these statistical pillars. It begins by deconstructing the fundamental Python code presented in your textbook, explaining each command as if it were being read aloud.³ From there, it builds a deeper understanding of

when and, more importantly, *why* an engineer chooses one measure over another. Finally, these concepts are applied to the sophisticated systems central to a mechatronics education. It will become clear how a simple concept like an "average" value is the bedrock of advanced applications, from industrial PID control loops that regulate manufacturing processes to the powerful Kalman filters that guide autonomous robots.⁴ This journey will demonstrate that to command the machine, one must first command the data.

Chapter 1: The Three Pillars of Centrality - Deconstructing the Code

Section 1.1: Setting the Scene - The grades List

Let us begin by verbally walking through the first line of code from the interactive Python session in your textbook. The command is: `grades = .` This statement performs two actions. First, it creates a list, which is a fundamental data structure in Python for storing an ordered collection of items. In this case, the items are the integer numbers eighty-five, ninety-three, forty-five, eighty-nine, and eighty-five. Second, it uses the equals sign, the assignment operator, to give this list a name, or variable: `grades`. This simple list now represents our sample dataset, a small collection of numerical scores that will serve as our laboratory for exploring the core concepts of central tendency.³

Section 1.2: The Manual Calculation - Mean as a Ratio

The next line of code demonstrates the foundational definition of the mean, or average, by calculating it "by hand." The expression is `sum(grades) / len(grades)`. This single line elegantly captures the mathematical formula for the mean, $\bar{x} = \frac{\sum x_i}{n}$.⁶ The

`sum()` function is a powerful built-in tool that iterates through every item in the `grades` list and adds them together, producing a single total value of 397. The `len()` function, short for length, is another built-in tool that simply counts how many items are in the list, which in this case is five. By dividing the total sum, 397, by the count, five, Python calculates the arithmetic average. The output for this operation is 79.4. This manual step is crucial for building intuition; before relying on specialized tools, it is vital to understand the principle they operate on.³

Section 1.3: The Power of Libraries - The `statistics` Module

While manual calculation is instructive, modern programming emphasizes efficiency and accuracy by leveraging pre-built toolkits known as libraries or modules. The line import statistics is a directive to the Python interpreter. It says, "I need to access a specialized set of tools for performing statistical calculations; please make them available to me".³ This practice of importing modules is a cornerstone of modern software development, allowing engineers to build upon the robust and tested work of others.

Once the module is imported, its functions can be accessed using "dot notation." The subsequent lines of code demonstrate this: statistics.mean(grades), statistics.median(grades), and statistics.mode(grades). The structure is module_name.function_name. Each of these functions takes an *iterable*—an object that can be looped over, like our grades list—as its input, or argument. The statistics.mean function performs the same calculation as the manual method, returning 79.4. The statistics.median function calculates the middle value, returning 85. And the statistics.mode function finds the most frequently occurring value, also returning 85.³

Section 1.4: Building Intuition - Sorting for Clarity

To understand *why* the median and mode are both 85, the textbook uses another helpful built-in function. The command sorted(grades) takes the original grades list and returns a new list where the elements have been arranged in ascending order. The output is the list: ``.³. This sorted view provides immediate visual clarity.

With the data ordered, the median becomes obvious. The median is defined as the value that separates the higher half from the lower half of a data sample. Since there are five values in the list—an odd number—the median is simply the value in the exact middle position, which is the third element: 85.⁷ If the list had an even number of values, as in the "Self Check" example

values = , the median is calculated by taking the average of the two middle values. When sorted, that list becomes ``. The two middle values are 84 and 88, and their average is \$ (84 + 88) / 2 \$, which is 86.0, the median reported in the example.³

The sorted list also makes the mode easy to identify. The mode is the value that appears most frequently in a dataset.⁹ Looking at the sorted list ``, it is clear that the number 85 appears twice, while all other numbers appear only once. Therefore, 85 is the mode.³

Section 1.5: When the Mode Fails - The Bimodal Clue

The textbook provides a crucial warning: the `statistics.mode` function will raise a `StatisticsError` if it is given a list like ``. In this dataset, both 85 and 93 appear twice, meaning there are two "most frequent" values. Such a dataset is described as *bimodal*.³ This is far more than a simple programming error to be worked around; for an engineer, it is a potent analytical signal.

In an engineering context, data is not just a collection of numbers; it is a representation of a physical process.¹⁰ A dataset that is supposed to describe a single, stable process—like the diameter of a part from a specific machine—is expected to have one central peak, or be unimodal. The emergence of two distinct peaks, a bimodal distribution, is a strong indicator that the underlying assumption of a single process is flawed.¹¹ The

`StatisticsError` from the Python function, therefore, acts as a red flag. It forces the engineer to stop and ask a critical question: "What two different phenomena are contributing to my data?" This transforms a programming inconvenience into a gateway for deep system analysis, a concept that will be explored further in Chapter 3.

Chapter 2: The Engineer's Dilemma - Choosing Your Measure

Section 2.1: The Mean - A System's Center of Gravity

The mean, or average, can be thought of as the "balance point" or "center of mass" for a set of data.¹² Every single data point contributes to its calculation; each value pulls on the final average, giving it "weight." This is both its greatest strength and its most significant vulnerability. The mean is the ideal measure of central tendency when the data is symmetrically distributed, such as in a classic bell curve, and is free from extreme values, known as outliers.⁶

Consider the process of quality control in manufacturing. A high-precision lathe produces a batch of cylindrical shafts. Due to microscopic vibrations, tool wear, and material inconsistencies, there will be tiny, random variations in the diameter of each shaft. If these variations are truly random, a histogram of the diameters will form a symmetrical, bell-shaped curve. In this scenario, the mean provides the best possible estimate of the true average

diameter for the entire batch. It incorporates all the small variations to give a single, representative value.⁷

Section 2.2: The Median - A Robust Anchor in a Storm

In stark contrast to the mean, the median is defined by its position. It is the 50th percentile, the value that sits squarely in the middle of the data when it is sorted, splitting the dataset into two equal halves.⁸ Its defining characteristic is its robustness, or its resistance to being influenced by outliers.⁷

To illustrate this, a common analogy is the analysis of housing prices in a neighborhood.⁶ Imagine a street with ten houses, all valued between \$300,000 and \$400,000. The mean and median prices would both be around \$350,000, accurately reflecting a typical home's value. Now, suppose a billionaire purchases a lot and builds a \$50 million mansion. The

mean house price on the street would skyrocket, pulled dramatically upward by this single extreme value. The new mean might be over \$5 million, a figure that gives a completely misleading picture of the neighborhood. The *median*, however, would be largely unaffected. When the new list of eleven house prices is sorted, the middle value would still be close to the original \$350,000, providing a much more faithful representation of what a typical house on that street costs.

Section 2.3: Outliers - The Ghosts in the Machine

An outlier is a data point that deviates so significantly from other observations that it arouses suspicion that it was generated by a different mechanism.¹⁴ For an engineer, understanding the source of an outlier is a critical diagnostic task. Outliers can arise from mundane sources, such as a manual data entry error or a temporary sensor malfunction.¹⁶ They can also represent rare but critically important real-world events, like a sudden pressure spike in a chemical reactor, a fraudulent financial transaction, or a network intrusion attempt.¹⁴

The crucial task for an engineer is not merely to detect an outlier but to correctly diagnose its origin. This diagnosis determines the appropriate action. If an outlier is confirmed to be noise or an error, it should be filtered out or its influence minimized by using a robust statistic like the median.¹⁴ However, if the outlier represents a genuine, critical event, it must be acted

upon immediately.

Consider a temperature sensor monitoring an industrial furnace. A sudden, extreme reading could be a faulty signal caused by electromagnetic interference—an error. Or, it could be the first indication of a dangerous thermal runaway event—a real emergency.¹⁸ Applying a median filter to the sensor data would effectively ignore the faulty signal, promoting system stability. But that same filter would also ignore the fire alarm, with potentially catastrophic consequences. Conversely, treating every noisy spike as a real emergency would lead to constant false alarms and an unstable, unusable system. This reveals that the choice between mean and median is not a simple statistical decision. It is a complex, context-dependent engineering judgment about the nature of the physical world the data represents.

Section 2.4: Cross-Industry Analogy - API Response Times

To solidify the distinction between mean and median, it is useful to examine a parallel problem from a different field: software engineering, specifically the monitoring of Application Programming Interface (API) performance. An API is a gateway that allows different software systems to communicate. When a user clicks a button in a mobile app, it might send a request to a server via an API to fetch data. The time it takes for the server to respond is a critical performance metric.¹⁹

If an engineering team measures the *mean* response time, their metric can be heavily skewed by a few unusually slow requests. Perhaps the network was congested for a moment, or a database query took an exceptionally long time for one or two users. These outliers can inflate the average, making the API's performance appear worse than what the majority of users actually experience.²¹

For this reason, performance engineers often focus on the *median* response time, also known as the 50th percentile or P50.²² The median provides a much better sense of the "typical" user's experience because it is not affected by those few extreme delays.¹⁹ The mean is still useful, as it reflects the total computational load on the server—every request, fast or slow, consumes resources. However, when the goal is to understand the experience of a representative user, the median is the more honest and reliable metric.²³ This reinforces the core principle: use the mean when the total or cumulative effect matters, but use the median when seeking a typical value in the presence of outliers.

Chapter 3: From Theory to the Factory Floor -

Applications in Mechatronics and Robotics

Section 3.1: Sensor Data Filtering - The First Line of Defense

Sensors are the senses of a robot, but the real world is a noisy place. An ultrasonic distance sensor on an autonomous mobile robot, for example, might occasionally return a wildly incorrect value—a radical outlier. This can happen if the sound wave is absorbed by a soft surface or creates a strange echo.²⁴ If the robot's control algorithm were to calculate the

mean of its last few distance readings, this single faulty measurement would drastically pull the average, potentially causing the robot to perceive a "phantom" obstacle and stop or swerve unnecessarily.

This is a classic scenario where the *median* is the superior tool. By taking the median of a small window of recent sensor readings (a technique known as a median filter), the sporadic outlier is effectively discarded. The robot's perception of distance remains stable and reliable, reflecting the true state of its environment. The median filter is therefore an essential first line of defense for any sensor known to be prone to occasional, large-magnitude errors.²⁴

In contrast, consider a different type of sensor, such as a temperature probe in a stable environment. This sensor might not produce extreme outliers, but its readings may have a small amount of consistent, random noise, causing them to fluctuate slightly around the true value. In this case, there are no extreme values to skew the result, making the *mean* an excellent choice. A *moving average*, which continuously calculates the mean of a sliding window of the most recent readings, is a highly effective technique for smoothing out this kind of noise and producing a stable, accurate temperature value.²⁴

Section 3.2: Bimodal Distributions - The Process Detective

Let us return to the `StatisticsError` encountered in Chapter 1, which signaled a bimodal dataset. In a manufacturing or quality control setting, the appearance of a bimodal distribution is a powerful diagnostic clue that an engineer can use to investigate the health of a process.¹⁰

Imagine a factory producing high-precision metal shafts. A quality control engineer measures

the diameter of hundreds of shafts from a production run and plots the results in a histogram. Instead of a single, bell-shaped peak, the histogram shows two distinct peaks. This bimodal distribution immediately indicates that the parts are not coming from a single, consistent process. The data is telling a story of two separate populations mixed together.¹¹ The cause could be one of several things: perhaps two different machines are producing the same part, but one is calibrated slightly differently than the other. Or maybe there are two shifts of operators with different training, or two separate batches of raw material with slightly different properties.¹⁰

The discovery of bimodality, whether through a Python error or data visualization, provides an immediate and actionable insight. It tells the engineer exactly where to focus their investigation to improve process consistency and product quality. It transforms a statistical anomaly into a clear path for engineering improvement.¹¹

Section 3.3: The Mean in Control Systems - The Heart of the PID Controller

The Proportional-Integral-Derivative (PID) controller is a workhorse of industrial automation, a fundamental algorithm used to regulate countless processes, from maintaining the temperature in an oven to controlling the speed of an electric motor.⁴ Its operation is a perfect example of the mean in action.

A PID controller functions within a closed loop. It continuously measures a "Process Variable" (PV) using a sensor—for instance, the current temperature of a chemical reactor. It then compares this measured value to the desired "Setpoint" (SP)—the target temperature the process should maintain. The difference between the setpoint and the process variable is the "error".²⁷ The controller's entire purpose is to calculate an output—such as the amount of power to send to a heating element—that will drive this error to zero.⁴

Crucially, the Process Variable fed into the controller is almost never a single raw sensor reading. Raw sensor data is often too noisy for direct use in a sensitive control loop. Instead, the PV is typically a filtered or averaged value, often calculated using a moving average. This use of the mean provides a stable, reliable signal of the system's current state. The controller then works tirelessly to adjust its output to make this *mean* value of the temperature precisely match the setpoint. The simple concept of an average is therefore fundamental to the stability and performance of a vast number of automated systems in modern industry.

Section 3.4: The Mean in State Estimation - The Brain of the Kalman Filter

For a more advanced application, consider the Kalman filter, a cornerstone algorithm in robotics, autonomous vehicles, and aerospace engineering.⁵ It is a powerful method for sensor fusion—the process of intelligently combining data from multiple, imperfect sensors to arrive at an estimate of a system's state that is more accurate than any single sensor could provide. A common example is fusing noisy data from a GPS receiver with noisy data from an Inertial Measurement Unit (IMU) to get a highly accurate estimate of a vehicle's position and velocity.³⁰

A Kalman filter approaches this problem from a probabilistic perspective. It does not represent the robot's position as a single, certain point. Instead, it represents the position as a Gaussian probability distribution—a bell curve. This distribution is defined by two key parameters: a *mean* and a *covariance*.⁵ The covariance is a measure of the uncertainty or spread of the distribution, while the

mean represents the system's most likely state—its best estimate of the true position.

The algorithm operates in a recursive "predict-then-update" cycle. First, it uses a model of the system's dynamics (e.g., laws of motion) to *predict* where the mean of the state will be at the next moment in time. Then, when a new sensor measurement arrives, it performs an *update* step, correcting the predicted mean based on the new information. This update is essentially a sophisticated weighted average, giving more weight to estimates (both the prediction and the new measurement) that have lower uncertainty.⁵ This demonstrates how the simple concept of a mean is scaled up to become the core component of one of the most powerful estimation algorithms in modern engineering.

Section 3.5: The Mode in Robotic Interaction - Categorizing the World

While the mean and median are suited for continuous numerical data, the mode excels when data is categorical or falls into a set of discrete values.²⁴ For a mechatronics engineer, this is useful for classification and decision-making tasks.

A straightforward example is a robot on an assembly line equipped with a vision system that classifies parts as they pass by. Due to lighting variations or odd angles, the vision system might occasionally misclassify a part. To make a robust decision, the robot's control system could look at the last ten classifications and calculate the mode. This would reveal the most

common part type it has seen recently, effectively filtering out any sporadic classification errors.

This concept extends to the cutting edge of robotics research, particularly in learning from demonstration. Researchers are developing methods for robots to understand "interaction modes" with objects in their environment.³³ For instance, when a robot observes a cabinet door, there are several distinct, discrete ways to interact with it: 'pulling it open', 'pushing it closed', 'sliding it sideways'. These are the modes of interaction. By analyzing human demonstration data, a machine learning model can identify these distinct clusters of behavior. The center of each cluster represents a mode—a frequent and meaningful way of interacting with the object. This allows the robot to sample its actions from a small set of high-probability, discrete options rather than trying to generate behavior from scratch. This connects the simple statistical concept of the mode directly to the advanced challenge of creating intelligent and adaptable robotic systems.³³

Chapter 4: The Functional Perspective - Understanding Reductions

Section 4.1: Demystifying Functional Programming

Your textbook makes an important note that the sum and len functions are examples of "functional-style programming reductions".³ To fully appreciate this, it is helpful to understand the core idea behind the functional programming paradigm. At its heart, functional programming treats computation as the evaluation of pure mathematical functions.³⁵ It emphasizes avoiding "side effects"—that is, modifying data outside of a function's scope or changing a system's state. A key element is the use of

pure functions, which are deterministic: for a given input, they will always produce the same output, regardless of when or where they are called.³⁶

Section 4.2: The Concept of Reduction (or Folding)

A *reduction*, also known as a *fold*, is a fundamental operation in the functional programming paradigm. It describes the process of taking a collection of values, such as a list, and applying a function cumulatively to its items in order to boil it down to a single, summary value.³⁹

The `sum()` function is a perfect illustration. Given the list `[1, 2, 3, 4, 5]`, `sum()` performs a reduction. It takes this collection of five numbers and reduces it to the single cumulative value of 397. Likewise, `len()` is also a reduction. It takes the same list of five numbers and reduces it to the single summary value of 5, which is the count of the items.³ Other built-in Python functions like

`min()` and `max()` are also reductions, as they take an iterable and return a single value. While these functions are highly optimized and readable for their specific tasks, Python also provides a general-purpose tool for this operation in the `functools` module, appropriately named `reduce()`.³⁷

Section 4.3: Why This Matters for an Engineer

Understanding the abstract computer science concept of "reduction" provides a powerful mental model for engineering data processing. An engineer's primary task is often to take vast, overwhelming streams of raw data from sensors and distill them into concise, meaningful, and actionable information.²⁵ This process of distillation

is a reduction.

For example, a temperature sensor in a control system may generate thousands of readings per minute. To make a control decision, this stream of data must be reduced to a single value—the mean temperature—which can then be fed into a PID controller.⁴ A camera on a production line captures millions of pixels per frame, but this data must be reduced to a single value, such as the count of defective products found. Thinking about data analysis as a pipeline of reductions encourages the design of clean, modular, and understandable data processing workflows. It frames the problem not as a series of complex loops and state changes, but as a flow where large quantities of data are systematically summarized into the critical single values needed for visualization, control, and decision-making.

Conclusion: The Data-Driven Engineer

This exploration began with a simple list of grades in a Python script and expanded to show how the underlying statistical concepts—mean, median, and mode—are the essential tools that power complex mechatronic systems. The journey has revealed that the mean acts as a system's center of gravity, a critical value for the precise calculations needed in control loops and state estimation algorithms. The median stands as a robust anchor, indispensable for filtering the noisy and unpredictable data that comes from sensors operating in the real world. The mode serves as a tool for categorization, helping machines to identify the most common state or classify discrete observations.

The most vital takeaway, however, is that for an engineer, the choice between these measures is never purely mathematical. It is an act of interpretation, deeply rooted in an understanding of the physical system that the data represents. A StatisticsError in a program is not a bug to be fixed but a clue to be investigated. An outlier in a dataset is not just a number to be discarded but a potential signal of a critical system event or an impending failure. As a data-driven engineer, your most important skill is not simply the ability to calculate these values, but the wisdom to interpret what they reveal about the state of your machine and its interaction with the world. Mastering this skill—the art of translating data into physical insight—is fundamental to designing the intelligent, robust, and reliable mechatronic systems of the future.

Works cited

1. Real life applications of Mechatronics | Capitol Technology University, accessed September 26, 2025,
<https://www.captechu.edu/blog/real-life-applications-of-mechatronics>
2. Hybrid Mode Sensor Fusion for Accurate Robot Positioning - PMC, accessed September 26, 2025, <https://pmc.ncbi.nlm.nih.gov/articles/PMC12115087/>
3. ch3 part 10.docx
4. The PID Controller & Theory Explained - NI - National Instruments, accessed September 26, 2025,
<https://www.ni.com/en/shop/labview/pid-theory-explained.html>
5. Kalman filter - Wikipedia, accessed September 26, 2025,
https://en.wikipedia.org/wiki/Kalman_filter
6. When to Use Mean vs. Median (With Examples) - Statology, accessed September 26, 2025, <https://www.statology.org/when-to-use-mean-vs-median/>
7. When to use mean vs median - Data Science Stack Exchange, accessed September 26, 2025,
<https://datascience.stackexchange.com/questions/46744/when-to-use-mean-vs-median>
8. Means and Medians: When To Use Which - Research Collective, accessed September 26, 2025,
<https://research-collective.com/means-and-medians-when-to-use-which/>
9. Real-Life Applications of Mean and Mode - GeeksforGeeks, accessed September 26, 2025,

- <https://www.geeksforgeeks.org/mathematics/real-life-applications-of-mean-and-mode/>
- 10. Bimodal Distribution - Six Sigma Study Guide, accessed September 26, 2025,
<https://sixsigmastudyguide.com/bimodal-distribution/>
 - 11. Bimodal and Unimodal Distributions in Six Sigma for Statistical ..., accessed September 26, 2025,
<https://www.6sigma.us/six-sigma-in-focus/bimodal-and-unimodal/>
 - 12. Mean vs. Median: Knowing the Difference - DataCamp, accessed September 26, 2025, <https://www.datacamp.com/tutorial/mean-vs-median>
 - 13. Real-world Applications of Mean, Median, and Mode - Intellspot, accessed September 26, 2025,
<https://www.intellspot.com/real-world-applications-of-mean-median-and-mode/>
 - 14. Outlier detection is an essential technique in data science used to identify data points that... | by Adnan Mazraeh | Medium, accessed September 26, 2025,
<https://medium.com/@adnan.mazraeh1993/outlier-detection-is-an-essential-technique-in-data-science-used-to-identify-data-points-that-dc91b10854bc>
 - 15. Why Detecting Outliers is Crucial for Accurate Data Analysis?, accessed September 26, 2025,
<https://www.dasca.org/newsroom/why-detecting-outliers-is-crucial-for-accurate-data-analysis>
 - 16. Handling Outliers|| Feature Engineering || Machine Learning - DEV Community, accessed September 26, 2025,
<https://dev.to/ngneha09/handling-outliers-feature-engineering-machine-learning-3316>
 - 17. (PDF) Outlier Detection Scheme to Handle Wireless Sensor Data - ResearchGate, accessed September 26, 2025,
https://www.researchgate.net/publication/394024998_Outlier_Detection_Scheme_to_Handle_Wireless_Sensor_Data
 - 18. A Survey of Outlier Detection Techniques in IoT: Review and ... - MDPI, accessed September 26, 2025, <https://www.mdpi.com/2224-2708/11/1/4>
 - 19. API Performance Monitoring—Key Metrics and Best Practices - Catchpoint, accessed September 26, 2025,
<https://www.catchpoint.com/api-monitoring-tools/api-performance-monitoring>
 - 20. What is a Good API Response Time? - Odown, accessed September 26, 2025,
<https://odown.com/blog/what-is-a-good-api-response-time/>
 - 21. Median v. Mean v. Total response time - WebTuna, accessed September 26, 2025,
<https://www.webtuna.com/median-v-mean-v-total-response-time/>
 - 22. Monitor API metrics - Atlassian Developer, accessed September 26, 2025,
<https://developer.atlassian.com/platform/forge/monitor-api-metrics/>
 - 23. Is median better than mean for reporting on website speed - Cross Validated, accessed September 26, 2025,
<https://stats.stackexchange.com/questions/80009/is-median-better-than-mean-for-reporting-on-website-speed>
 - 24. Mean, Median and Mode for sensor data. - Parallax Forums, accessed September 26, 2025,
<https://forums.parallax.com/discussion/167129/mean-median-and-mode-for-sens>

or-data

25. Best Practices for Processing and Analyzing Robotics Data - Foxglove, accessed September 26, 2025,
<https://foxglove.dev/blog/best-practices-for-processing-and-analyzing-robotics-data>
26. Proportional–integral–derivative controller - Wikipedia, accessed September 26, 2025,
https://en.wikipedia.org/wiki/Proportional%E2%80%93integral%E2%80%93derivative_controller
27. Help Tuning PID controller : r/AskEngineers - Reddit, accessed September 26, 2025,
https://www.reddit.com/r/AskEngineers/comments/21kfz7/help_tuning_pid_controller/
28. Principles of PID Control and Tuning | Eurotherm Limited, accessed September 26, 2025,
<https://www.eurotherm.com/us/temperature-control-us/principles-of-pid-control-and-tuning/>
29. Using Kalman Filters for Sensor Fusion in Robotics - Patsnap Eureka, accessed September 26, 2025,
<https://eureka.patsnap.com/article/using-kalman-filters-for-sensor-fusion-in-robotics>
30. 18 Sensor Fusion - Autonomous Systems Laboratory, accessed September 26, 2025,
https://stansfordasl.github.io/PoRA-I/aa274a_aut2223/pdfs/notes/lecture16and17.pdf
31. Implementing a Kalman Filter in Python for Sensor Fusion - ThinkRobotics.com, accessed September 26, 2025,
<https://thinkrobotics.com/blogs/learn/learn-to-design-and-3d-print-robotic-gripers-discover-materials-mechanisms-sensors-and-optimization-techniques-for-effective-robotic-arm-end-effectors>
32. Sensor Fusion With Kalman Filter. Introduction | by Satya - Medium, accessed September 26, 2025,
https://medium.com/@satya15july_11937/sensor-fusion-with-kalman-filter-c648d6ec2ec2
33. Discovering Robotic Interaction Modes with Discrete Representation Learning - arXiv, accessed September 26, 2025, <https://arxiv.org/html/2410.20258v1>
34. Continuous or Discrete - Robotics Stack Exchange, accessed September 26, 2025, <https://robotics.stackexchange.com/questions/2148/continuous-or-discrete>
35. Functional Programming in Python - GeeksforGeeks, accessed September 26, 2025,
<https://www.geeksforgeeks.org/python/functional-programming-in-python/>
36. Functional Programming HOWTO — Python 3.13.7 documentation, accessed September 26, 2025, <https://docs.python.org/3/howto/functional.html>
37. Functional Programming in Python: When and How to Use It - Real Python, accessed September 26, 2025,

<https://realpython.com/python-functional-programming/>

38. The power of Python Map, Reduce and Filter - Functional Programming for Data Science, accessed September 26, 2025,
<https://www.analyticsvidhya.com/blog/2021/09/the-power-of-python-map-reduce-and-filter-functional-programming-for-data-science/>
39. Python's reduce(): From Functional to Pythonic Style – Real Python, accessed September 26, 2025, <https://realpython.com/python-reduce-function/>