Comparative Performance Analysis of GPU and CPU Processing for Digital Filtering Operations on Images Captured by the NAO Humanoid Robot

Vitor Amadeu Souza [1]

[1] Department of Electrical Engineering
University Center of Volta Redonda (UniFOA)
Volta Redonda, RJ, Brazil
vitor.amadeu@foa.org.br
https://orcid.org/0009-0002-1857-6799

Abstract. *This work presents a quantitative comparison of GPU and CPU performance for digital image filtering operations, using the NAO humanoid robot as the image source. Three algorithms were evaluated: Gaussian filtering, Sobel filtering, and FFT-based high-pass filtering, with experiments varying batch size (1 to 50 images) and resolution (320×240 to 1280×960 pixels). Results show that GPU advantage is operation-dependent: FFT achieved speedups of up to 63.26× at high resolution, while Gaussian filtering only outperformed CPU for larger batch sizes. Operation-specific efficiency thresholds were established, providing practical guidelines for architecture selection in robotic computer vision systems.*

1. Introduction

The evolution of computer vision systems in robotics has increasingly demanded computational power for real-time processing of complex visual data. Traditionally, these operations were executed on central processing units (CPUs). However, the advent of general-purpose graphics processing units (GPUs) has revolutionized the parallel computing paradigm in scientific and engineering applications [Kirk e Hwu 2016]. The massively parallel architecture of GPUs, originally designed for graphics rendering, has proven highly effective for operations that can be decomposed into multiple independent tasks executed simultaneously [Sanders e Kandrot 2010].

In the context of humanoid robotics, platforms such as the SoftBank Robotics NAO robot, have been widely adopted in academic research due to their versatility and compatibility with various development frameworks [Gouaillier et al. 2009]. These robots incorporate stereo vision systems that generate large volumes of visual data, which require efficient processing for navigation, object recognition, and human-robot interaction tasks [Lowe 2004].

Digital filtering operations constitute essential elements in image processing pipelines, being employed for enhancement, smoothing, and feature extraction [Gonzalez e Woods 2017]. Among these, Gaussian filters are used for noise reduction and smoothing, Sobel filters for edge detection via gradient computation, and the fast Fourier transform (FFT) for frequency-domain filtering [Brigham 1988]. The computational efficiency of these algorithms is critical in applications demanding real-time response.

GPU-based parallel computing relies on NVIDIA's CUDA (Compute Unified Device Architecture) model, which allows the execution of thousands of threads simultaneously through a Single Instruction Multiple Data (SIMD) architecture [Nickolls et al. 2008]. This approach is particularly suitable for matrix operations and convolutions commonly used in image processing algorithms. However, data transfer between main memory and GPU memory introduces latencies that can negatively affect performance for small data volumes [Harris 2007].

Previous studies have investigated the application of GPUs in image processing, demonstrating significant speedups for specific operations [Fialka e Cadik 2006, Podlozhnyuk 2007]. Nevertheless, few works have established quantitative criteria for choosing between CPU and GPU architectures based on operation type, batch size, and image resolution. This gap is especially relevant in robotic systems, where computational resources are limited, and architectural decisions must be supported by empirical evidence.

This work aims to address this gap through a systematic comparative analysis of CPU and GPU performance for digital filtering operations, establishing quantitative parameters that guide the selection of the most efficient architecture according to application-specific characteristics. The results provide practical contributions for the development of computer vision systems in robotics, where performance optimization is essential for operational feasibility.

The remainder of this paper is organized as follows. Section II presents the theoretical foundations of GPU parallel computing and digital filtering operations, including the mathematical formulations of Gaussian, Sobel, and FFT filters. Section III describes the NAO robot's camera specifications and hybrid system architecture used for image acquisition. Section IV details the experimental methodology, including the hardware configuration, implementation approach for both CPU and GPU architectures, and the benchmarking protocol. Section V formalizes the systematic evaluation through algorithmic representation of the benchmark pipeline. Section VI presents the experimental results, analyzing performance patterns across different operations, batch sizes, and resolutions, and identifying efficiency thresholds for architecture selection. Finally, Section VII concludes with the main findings and suggests future research directions.

2. Theoretical grounding

GPU parallel computing is based on exploiting massive parallelism through the simultaneous execution of thousands of threads organized in blocks and grids [Kirk e Hwu 2016]. This architecture fundamentally differs from multi-core CPUs, which prioritize the optimization of individual threads using techniques such as out-of-order execution, branch prediction, and complex cache hierarchies [Hennessy e Patterson 2017]. While modern CPUs typically feature 4 to 64 cores optimized for sequential execution, GPUs incorporate thousands of simple cores specialized in parallel computation [Owens et al. 2008].

NVIDIA's CUDA programming model allows the implementation of kernels executed by multiple threads in parallel [Nickolls et al. 2008]. Each thread performs the same instruction on different data, characterizing the SIMD (Single Instruction Multiple Data) model. The GPU memory hierarchy includes local registers, high-speed shared memory, global memory, and texture memory, each with specific latency and bandwidth

characteristics [Harris 2007].

Digital filtering operations in images are naturally suitable for parallel processing due to the independence between adjacent pixels in most algorithms. The Gaussian filter, defined as the convolution of the image with a two-dimensional Gaussian kernel, can be optimized through its separability property, allowing decomposition into two sequential one-dimensional convolutions [Young e Van Vliet 1995]. This optimization reduces computational complexity from O(n^2) to O(n) per pixel, where n is the kernel size.

The Sobel filter is a first-derivative operator used for edge detection through the computation of the image intensity gradient [Sobel e Feldman 1968]. Its parallel implementation benefits from the simultaneous application of horizontal and vertical kernels, followed by combination of the results using the Euclidean magnitude. GPU efficiency depends on optimized memory access and the use of shared memory to reduce redundant global memory accesses.

The fast Fourier transform (FFT) is one of the operations most benefited by parallel processing due to its inherently parallel algorithmic structure [Cooley e Tukey 1965]. GPU implementations of the FFT, using optimized libraries such as cuFFT, exploit multiple levels of parallelism: parallelism across different FFTs in a batch, parallelism within a single FFT through radix decomposition, and parallelism in applying frequency-domain filters [Volkov e Kazian 2008].

Frequency-domain filtering offers computational advantages for large convolution kernels, where the cost of forward and inverse transforms is offset by the efficiency of pointwise multiplication in the transformed domain [Brigham 1988]. For high-pass filters, the implementation consists of multiplying the image spectrum by a mask that attenuates low frequencies while preserving high frequencies associated with edges and details [Oppenheim e Schafer 2009].

The relative performance between CPU and GPU is influenced by several factors, including data transfer overhead, memory access patterns, utilization of hardware resources, and efficiency of the libraries used [Lee et al. 2010]. Data transfer between host and device introduces latencies that can outweigh speedup gains for small data volumes, establishing a minimum threshold for viable GPU processing [Che et al. 2009].

Optimizing CUDA kernels requires careful consideration of memory coalescing, multiprocessor occupancy, and load balancing among threads [Harris 2007]. Techniques such as memory coalescing, shared memory utilization, and minimizing branch divergence are essential to maximize performance [Volkov 2010]. Additionally, the use of optimized libraries such as GPU-accelerated OpenCV and PyTorch provides highly efficient implementations incorporating these low-level optimizations.

3. Cameras and Hybrid System

The NAO robot is equipped with two CMOS cameras, one positioned on the top and the other on the bottom, providing it with computer vision capabilities [Gouaillier et al. 2008]. The camera's field of view (FOV) has a diagonal angle of 47.64°, covering an effective area of 39.7° from the focal point. These technical parameters are fundamental for the development of efficient computer vision algorithms [Aldebaran Robotics 2021].

Accurate FOV measurements are essential for image processing and real-time object recognition, enabling proper calibration of detection and tracking algorithms [Ficht et al. 2018]. Figure 1 illustrates the NAO robot's field of view.

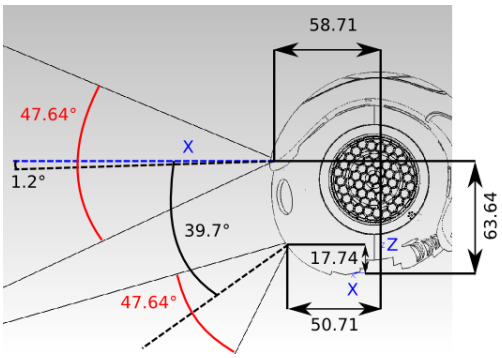

Figure 1. Field of view (FOV) of the NAO robot [Aldebaran Robotics 2021].

The internal architecture of NAO consists of multiple integrated modules that configure a hybrid system. Among these components, the CMOS camera stands out as it provides the visual data to be processed [Gouaillier et al. 2008]. This camera is connected to the main CPU board, which employs an Intel Atom E3845 processor assisted by an ARM-7 microcontroller. Communication with peripheral modules occurs through interfaces such as the I2C bus [Gouaillier et al. 2008].

Additionally, the robot includes dsPIC modules that control the motors and sensors distributed throughout the body, including the head, shoulders, elbows, hips, knees, and ankles. This hybrid architecture ensures coordinated execution of movements and complex interactions with the environment [Heremans et al. 2016]. Figure 2 illustrates the NAO robot's hybrid system.

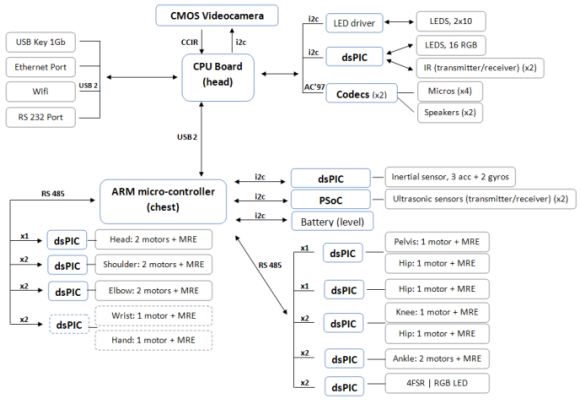

Figure 2. Hybrid system architecture of the NAO robot[Aldebaran Robotics 2021].

4. Methodological procedure

The experimental methodology was developed to systematically evaluate the comparative performance between CPU and GPU processing of digital image filtering operations. Considering that the NAO V6 robot only supports Python 2.7, remote processing was implemented using an off-board processing approach, in which a computer communicates

with the robot via WiFi. Consequently, the experimental system consisted of a NAO v6 humanoid robot connected to a workstation equipped with an Intel Core i9-12900 CPU, 64 GB DDR4 RAM, and an NVIDIA GeForce RTX 4070 GPU with 12 GB VRAM. The operating system used was Ubuntu 24.04 LTS with CUDA Toolkit 12.8 and PyTorch 2.8.

The experimental protocol involved real-time image acquisition through the NAO robot's camera system, using the NAOqi API to interface with the vision subsystem. Images were captured at a maximum resolution of 640×480 pixels at 15 fps using the top camera and subsequently resized to the test resolutions. The robot connection was established via TCP/IP using IP address 172.15.1.80 and the default port 9559.

Three filtering operations were implemented for both CPU and GPU: Gaussian filter for smoothing, Sobel filter for edge detection, and fast Fourier transform (FFT) with a high-pass filter for frequency enhancement. CPU implementations utilized OpenCV 4.12.0 for basic operations and SciPy 1.16.1 for FFT operations. GPU implementations were developed using PyTorch 2.8.0 with CUDA support, leveraging native optimizations for parallel matrix operations.

The Gaussian filter employed a 15×15 kernel with a separable implementation for maximum efficiency. The Sobel filter used standard 3×3 kernels for horizontal and vertical gradient computation, followed by combination using the Euclidean magnitude. The FFT operation incorporated a circular high-pass filter with a cutoff radius of 30 pixels at the center of the frequency spectrum.

The benchmarking protocol involved systematic variation of three experimental parameters: image resolution (320×240, 640×480, 1280×960 pixels), batch size (1, 5, 10, 20, 50 images), and type of filtering operation. Multiple iterations were executed for each parameter combination, with processing time measurements recorded.

Timing methodology included explicit synchronization with the GPU using `torch.cuda.synchronize()` to ensure that all parallel operations were completed before measurement. Times were measured exclusively for the processing operations, excluding data transfer between CPU and GPU to isolate pure computational costs.

For each experimental configuration, multiple batches were processed to obtain statistically significant averages. Speedup was calculated as the ratio between CPU and GPU processing times, with values greater than 1.0 indicating GPU advantage. Statistical analysis included calculation of means, standard deviations, and identification of outliers to ensure result reliability.

Implementation correctness was validated through pixel-wise comparison between CPU and GPU results, using root mean square error (RMSE) and structural similarity index (SSIM) metrics. All implementations showed numerical equivalence within floating-point precision, validating the consistency of algorithms across architectures.

5. Benchmark Algorithm

This section formalizes the systematic evaluation methodology through algorithmic representation, detailing the comparative performance measurement process between CPU and GPU architectures for digital filtering operations.

The algorithm 1 implements a three-level nested loop structure for systematic

Algorithm 1 GPU vs CPU Benchmark Pipeline

Input: NAO IP, resolutions, batch sizes, operations
Output: Performance metrics (speedup, times)

// Initialization
Connect to NAO at IP:9559
$device \leftarrow$ "cuda" if GPU available else "cpu"
$base\_frame \leftarrow$ CaptureFromNAO(camera="top")

// Benchmark Loop
for $resolution$ in $[(320, 240), (640, 480), (1280, 960)]$ **do**
  $resized\_frame \leftarrow$ Resize($base\_frame$, $resolution$)
  for $batch$ in $[1, 5, 10, 20, 50]$ **do**
    $images \leftarrow$ Replicate($resized\_frame$, $batch \times 3$)
    for $operation$ in [Gaussian, Sobel, FFT] **do**
      // CPU Benchmark
      $start_{cpu} \leftarrow$ GetTime()
      if $batch = 1$ **then**
        for img in $images$ **do**
          $\_ \leftarrow$ CPU_Filter(img, $operation$)
        end for
      else
        for b in CreateBatches($images$, $batch$) **do**
          $\_ \leftarrow$ CPU_BatchFilter(b, $operation$)
        end for
      end if
      $time_{cpu} \leftarrow$ (GetTime() $- start_{cpu}$) / num_batches

      // GPU Benchmark
      Synchronize GPU
      $start_{gpu} \leftarrow$ GetTime()
      $tensors \leftarrow$ ConvertToTensor($images$, $device$, $batch$)
      for t in $tensors$ **do**
        $\_ \leftarrow$ GPU_Filter(t, $operation$)
      end for
      Synchronize GPU
      $time_{gpu} \leftarrow$ (GetTime() $- start_{gpu}$) / num_batches

      // Compute Metrics
      $speedup \leftarrow time_{cpu}/time_{gpu}$
      $pixels \leftarrow resolution[0] \times resolution[1] \times batch$
      Store($resolution$, $batch$, $operation$, $time_{cpu}$, $time_{gpu}$, $speedup$, $pixels$)
    end for
  end for
end for

GenerateReport()

evaluation. The outermost loop (lines 10-43) iterates through test resolutions (320×240, 640×480, 1280×960 pixels), capturing and resizing the base frame obtained from NAO's camera (line 11). The intermediate loop (lines 12-42) varies batch sizes from 1 to 50 images, creating replicated datasets by tripling each batch size to enable multiple measurement iterations (line 13). The innermost loop (lines 14-41) evaluates the three filtering operations: Gaussian smoothing, Sobel edge detection, and FFT-based high-pass filtering.

CPU benchmarking (lines 17-27) distinguishes between single-image and batch processing modes. For single images (lines 19-21), each image is independently processed through the filter function. For batch processing (lines 23-25), images are grouped into batches of the specified size and processed collectively through optimized batch filter functions that leverage vectorization. Average processing time per batch is computed by dividing total elapsed time by the number of processed batches (line 27).

GPU benchmarking (lines 30-37) requires explicit synchronization before and after timing measurements to ensure accurate latency capture, accounting for GPU's asynchronous execution model. Images are converted to PyTorch tensors and transferred to GPU memory (line 32), with automatic batching for multi-image processing. GPU filter implementations exploit CUDA parallelism through PyTorch's optimized convolution operations (F.conv2d) for spatial filters and torch.fft operations for frequency-domain filtering (line 34). Post-execution synchronization (line 36) ensures all parallel operations complete before time recording.

Performance metrics computation (lines 40-42) calculates speedup as the ratio of CPU to GPU times, with values above 1.0 indicating GPU advantage. Total pixel count combines resolution and batch size to establish efficiency thresholds. All metrics are stored in a structured format for subsequent statistical analysis, including identification of optimal configurations and breakeven points where GPU processing becomes advantageous over CPU execution.

6. Results and commentary

The experimental results reveal distinct performance patterns between CPU and GPU architectures, which vary significantly depending on the type of operation, batch size, and image resolution. Quantitative analysis demonstrates that GPU relative efficiency is not uniform across different filtering operations, highlighting the need for specific criteria when selecting the appropriate architecture.

For fast Fourier transform (FFT) operations, results consistently demonstrate GPU superiority across nearly all tested configurations. The highest observed speedup was 63.26× for individual images at 1280×960 pixels, evidencing the suitability of the parallel architecture for FFT operations. This finding aligns with previous studies identifying FFT as a naturally parallelizable operation due to its divide-and-conquer algorithmic structure [Volkov e Kazian 2008]. For processing batches, FFT speedups remained consistently high, ranging from 18.3× to 62.7× depending on the configuration, demonstrating significant benefits from parallel processing regardless of batch size.

Sobel filter operations exhibited more heterogeneous behavior, with GPU efficiency primarily observed for high-resolution images and large batches. For individual

high-resolution images (1280×960), speedup reached 7.70×, whereas for batches of 50 images at the same resolution, speedup reached 17.93×. These results indicate that the Sobel filter benefits from both inter-pixel parallelism and parallelism across multiple simultaneously processed images. However, for low-resolution single images, the CPU outperformed the GPU, suggesting that GPU setup overhead outweighs parallelism benefits for small data volumes.

The Gaussian filter exhibited intermediate efficiency patterns, with moderate speedups ranging from 2.42× to 7.98× depending on configuration. Interestingly, this filter showed greater sensitivity to batch size than image resolution, with GPU efficiency consistently observed only from batches of 10 images. For individual image processing, speedups were modest (2.89× to 2.92×), indicating that the separable implementation of the Gaussian filter significantly reduces the advantage of massive GPU parallelism.

Efficiency threshold analysis reveals that different operations require distinct minimum data volumes to justify GPU processing. For Gaussian filters, the identified threshold was 307,200 pixels (equivalent to a 640×480 image), while Sobel filters demonstrated efficiency starting from 76,800 pixels (320×240). FFT operations exhibited the same threshold as the Gaussian filter but with substantially higher speedups above this point. These findings provide practical guidelines for computer vision systems, where processing architecture selection can be based on quantitative criteria.

Batch size trends revealed differentiated behavior between operations. FFT showed increasing speedups up to medium batch sizes (10–20 images), followed by stabilization, whereas spatial filters (Gaussian and Sobel) exhibited more linear growth in speedups with batch size increase. This difference suggests that FFT primarily benefits from intra-operation parallelism, while spatial filters exploit inter-image parallelism.

Scalability analysis with respect to image resolution indicates that all operations exhibit increasing speedups with higher resolutions, confirming that computationally intensive operations favor GPU parallel architecture. At the highest tested resolution (1280×960), all algorithms demonstrated speedups superior to lower resolutions, with particularly pronounced differences for FFT, which achieved a speedup of 15.12× at 640×480 compared to 63.26× at 1280×960 for individual images.

Results also show that GPU data transfer and setup overhead significantly impact performance for small data volumes. For single-image low-resolution batches, CPU performance often exceeded GPU performance, particularly for Gaussian filters, where GPU times were up to 65 times higher than CPU. This behavior underscores the importance of considering not only raw processing capability but also costs associated with setup and data transfer.

Figure 3 provides a visual summary of the experimental results from multiple analysis perspectives. The left panel demonstrates FFT dominance in terms of speedup, maintaining values consistently above 15× for all tested batch sizes at 640×480 resolution. An initial peak of approximately 59× for batches of 5 images is followed by stabilization between 42× and 47× for larger batches, indicating saturation of available parallel resources. Gaussian and Sobel filters show contrasting behaviors: Gaussian exhibits moderate and stable speedup growth with batch size (reaching approximately 6× for 50 images), while Sobel initially shows unfavorable behavior that gradually improves,

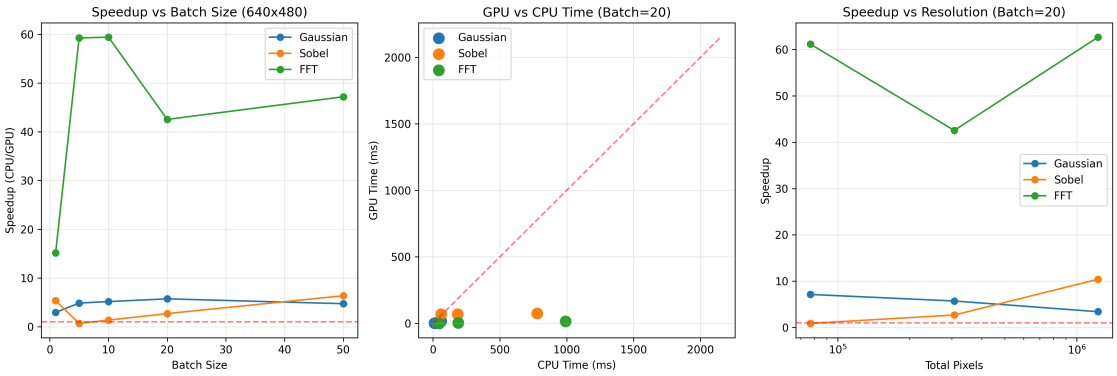

Figure 3. GPU vs CPU speedup analysis.

reaching 6.3× only for the largest batches.

The central graph illustrates the direct relationship between computational complexity and parallel processing benefit by plotting absolute GPU versus CPU times. Points below the red diagonal line (representing performance parity) confirm GPU superiority for the tested operations with batches of 20 images. Notably, FFT points demonstrate substantially reduced GPU times compared to CPU, quantitatively validating the observed high speedups.

Resolution-based scalability analysis, shown in the right graph, indicates that all operations benefit from increased image resolution, confirming that higher computational density favors GPU parallel architecture. FFT maintains dominant performance across resolutions, with speedup increasing from approximately 42× at 640×480 to 63× at 1280×960. Sobel exhibits pronounced growth, evolving from roughly 3× to 10× with resolution increase, while Gaussian remains relatively stable between 3× and 7×.

Variability across different runs was minimal, with coefficients of variation below 5% for most measurements, indicating high reproducibility and stability of the developed implementations. This consistency validates the robustness of the experimental methodology and the reliability of the reported speedups.

The technical repository for this work is available at: `https://github.com/vitor-souza-ime/cvcg`.

7. Conclusions

This study established quantitative criteria for selecting between CPU and GPU architectures for digital filtering operations in robotic computer vision systems. The results demonstrate that the relative efficiency between these architectures is highly dependent on the specific type of operation, the volume of data processed, and image characteristics, refuting generalized approaches that uniformly favor one architecture over the other.

The fast Fourier transform (FFT) emerged as the operation most benefited by parallel processing, consistently showing speedups above 15× and reaching peaks of 63.26× for high-resolution images. This result positions FFT as a priority candidate for GPU implementation in systems requiring frequency-domain filtering. In contrast, spatial filters such as Gaussian and Sobel exhibited more moderate and conditional benefits, requiring

minimum data volumes to justify the use of GPU resources.

The identified efficiency thresholds provide practical guidelines for the development of computer vision systems: FFT operations should preferably be executed on GPU regardless of data volume; Sobel filters benefit from GPU execution for resolutions above 320×240 pixels; and Gaussian filters require batches of at least 10 images or resolutions higher than 640×480 pixels to justify parallel processing.

The developed methodology demonstrates direct applicability in real robotic systems and was validated through image capture and processing from a NAO robot under operational conditions. The observed speedups indicate significant potential for performance improvement in real-time applications, particularly relevant for autonomous navigation and pattern recognition in mobile robotics.

The contributions of this work include establishing evidence-based quantitative criteria for processing architecture selection, identifying specific thresholds for different types of filtering operations, and demonstrating the feasibility of hybrid implementations that optimize computational resources through dynamic selection between CPU and GPU based on the characteristics of the processed data.

Future work may explore extending this methodology to more complex computer vision operations, such as SIFT/SURF feature detection, template matching, and optical flow algorithms. Additionally, investigating the applicability of the established criteria to other robotic platforms and GPU architectures represents a promising direction for generalizing these results. The implementation of automatic architecture selection systems based on the identified thresholds constitutes a concrete opportunity for practical application of the knowledge generated by this study.

References

D. B. Kirk and W. W. Hwu, *Programming Massively Parallel Processors: A Hands-On Approach*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann, 2016. [Online]. Available: https://www.elsevier.com/books/programming-massively-parallel-processors/kirk/978-0-12-811986-0

J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Boston, MA, USA: Addison-Wesley, 2010. [Online]. Available: https://developer.nvidia.com/cuda-example

D. Gouaillier *et al.*, "Mechatronic design of NAO humanoid," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, Kobe, Japan, 2009, pp. 769–774, doi: 10.1109/ROBOT.2009.5152516.

D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2004, doi: 10.1023/B:VISI.0000029664.99615.94.

R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. New York, NY, USA: Pearson, 2017. [Online]. Available: https://www.pearson.com/us/higher-education/program/Gonzalez-Digital-Image-Processing-4th-Edition/PGM241219.html

E. O. Brigham, *The Fast Fourier Transform and its Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

J. Nickolls *et al.*, "Scalable parallel programming with CUDA," *Queue*, vol. 6, no. 2, pp. 40–53, 2008, doi: 10.1145/1365490.1365500.

M. Harris, "Optimizing CUDA," in *SC07 Supercomputing Conf.*, Reno, NV, USA, 2007. [Online]. Available: https://developer.nvidia.com/blog/how-optimize-data-transfers-cuda-cc/

I. T. Young and L. J. Van Vliet, "Recursive implementation of the Gaussian filter," *Signal Processing*, vol. 44, no. 2, pp. 139–151, 1995.

I. Sobel and G. Feldman, "A 3x3 isotropic gradient operator for image processing," in *Pattern Classification and Scene Analysis*, pp. 271–272, 1968. [Online]. Available: https://www.researchgate.net/publication/239398674_An_Isotropic_3_3_Image_Gradient_Operator

J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, no. 90, pp. 297–301, 1965, doi: 10.1090/S0025-5718-1965-0178586-1.

V. Volkov and B. Kazian, "Fitting FFT onto the G80 architecture," Computer Science Division, University of California, Berkeley, Technical Report UCB/EECS-2008-132, 2008.

A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, 3rd ed. Boston, MA, USA: Pearson, 2009. [Online]. Available: https://www.pearson.com/us/higher-education/program/Oppenheim-Discrete-Time-Signal-Processing-3rd-Edition/PGM212808.html

V. W. Lee *et al.*, "Debunking the 100X GPU vs. CPU myth: An evaluation of throughput computing on CPU and GPU," *ACM SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 451–460, 2010, doi: 10.1145/1816038.1816021.

S. Che *et al.*, "Rodinia: A benchmark suite for heterogeneous computing," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Austin, TX, USA, 2009, pp. 44–54, doi: 10.1109/IISWC.2009.5306797.

V. Volkov, "Better performance at lower occupancy," in *GPU Technology Conference*, San Jose, CA, USA, 2010.

GOUAILLIER, David et al. The NAO humanoid: a combination of performance and affordability. *arXiv preprint arXiv:0807.3223*, 2008.

ALDEBARAN ROBOTICS. *NAO Technical Documentation*, Softbank Robotics, Technical Report, 2021. Disponível em: http://doc.aldebaran.com/2-8/home_nao.html. Acesso em: 2 set. 2025.

FICHT, Grzegorz et al. NimbRo-OP2X: Adult-sized open-source 3D printed humanoid robot. In: *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, IEEE, 2018. p. 1-9.

HEREMANS, François et al. Bio-inspired balance controller for a humanoid robot. In: *2016 6th IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, IEEE, 2016. p. 441-448.

O. Fialka and M. Cadik, "FFT and convolution performance in image filtering on GPU," in *Proc. 10th Int. Conf. Information Visualisation (IV)*, London, U.K., 2006, pp. 609–614, doi: 10.1109/IV.2006.53.

V. Podlozhnyuk, "Image convolution with CUDA," NVIDIA Corporation, White Paper, 2007.

J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 6th ed. San Francisco, CA, USA: Morgan Kaufmann, 2017.

J. D. Owens *et al.*, "GPU computing," *Proc. IEEE*, vol. 96, no. 5, pp. 879–899, 2008, doi: 10.1109/JPROC.2008.917757.