Comprehensive Technical Report on CUDA-based Sensor Fusion for Autonomous Vehicles

Emma Berry

Dr. Ogunfunmi

ECEN 331

18 March 2024

Abstract

This detailed report examines an implementation of Sensor Fusion, aimed at enhancing Autonomous Vehicle (AV) technology through the integration of GPS, LIDAR, and camera data using CUDA. The project's ambition was to transcend the limitations of traditional CPU-based sensor fusion methods by leveraging the parallel computing capabilities of CUDA, thereby significantly improving the real-time processing of sensor data. This document not only outlines the technical framework and methodologies employed but also provides an in-depth analysis of the challenges faced and the solutions adopted to overcome them.

Introduction

The advent of AV technology has necessitated the development of advanced sensor fusion techniques capable of real-time processing and interpretation of vast amounts of data. Traditional approaches, while effective to a degree, have been hampered by processing speed limitations inherent to CPU-based computations. This project was conceived to address these limitations by transitioning to a CUDA-based approach, thereby harnessing the power of parallel computing to enhance data processing efficiency and throughput. The integration of GPS and video support further augments the system's capabilities, providing a comprehensive and accurate real-time view of the vehicle's surroundings.

Background

Autonomous vehicles rely on a myriad of sensors to navigate and understand their environment. Sensor fusion involves combining data from these diverse sources, such as cameras and LIDAR, to create a cohesive and detailed representation of the vehicle's surroundings. The project's aim was to enhance this fusion process by implementing a CUDA-based system capable of handling the computational demands of real-time data processing.

Objectives

The primary objectives of the project were as follows:

- Transition the existing sensor fusion algorithm from C++ to CUDA C++.
- Integrate real-time GPS data to enhance positional accuracy.
- Incorporate video support to improve the visualization of fused sensor data.
- Optimize the CUDA implementation to achieve superior performance compared to the CPU-based approach.

Methods

Code Structure and Workflow

The project's codebase is structured around key components designed to process and integrate data from LIDAR, GPS, and camera sensors. At the heart of this structure is the CUDA kernel, transformLidarPoints, responsible for converting 3D LIDAR points to 2D image points. This kernel employs shared memory and matrix multiplication techniques to achieve efficient parallel processing. Calibration data is crucial for accurate transformations and is managed to ensure compatibility with GPU processing.

The overlayGPS function overlays GPS data onto the video feed, enhancing the contextual information provided by the sensor fusion system. GPS data is read and processed in real-time, ensuring accurate positional information is reflected in the system's output.

Camera images are processed using OpenCV, with CUDA extensions utilized for GPU-accelerated operations. The comprehensive workflow encapsulated by the projectLidarToCamera function includes reading sensor data, processing it on the GPU, and overlaying the processed data onto camera images for display.

Execution Plan

The execution plan was divided into four main phases:

- CUDA Conversion: This phase involved rewriting the existing C++ algorithm for CUDA C++, emphasizing parallel processing to offload computational tasks from the CPU to the GPU.
- GPS Integration: Implementing GPS sensor support entailed developing a system for reading and processing real-time GPS data, thereby enhancing the positional accuracy of the sensor fusion output.
- Video Support: Video support was added to enable the real-time visualization of fused sensor data, providing a dynamic and comprehensive view of the vehicle's environment.
- Optimization: The final phase focused on optimizing the CUDA code and data transfer processes to maximize the system's performance and efficiency.

Results

The transition to a CUDA-based system yielded significant improvements in processing speed and throughput, with the project achieving approximately 2.5 times the performance of its CPU-based predecessor. The video feed demonstrated smooth operation, with LIDAR overlays maintaining relative accuracy. While the CUDA implementation lacked a color gradient in the LIDAR overlay, GPS data integration was precise, enhancing the system's utility. Screenshots of the initial and final Sensor Fusion outputs can be found on the next page.

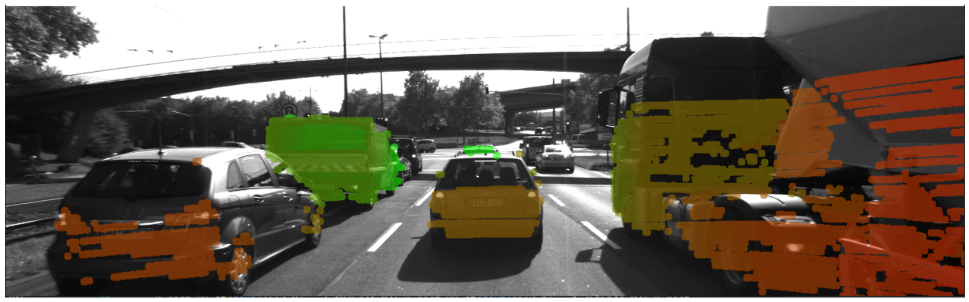
Figure 1: Pre-Existing Sensor Fusion Output - static image

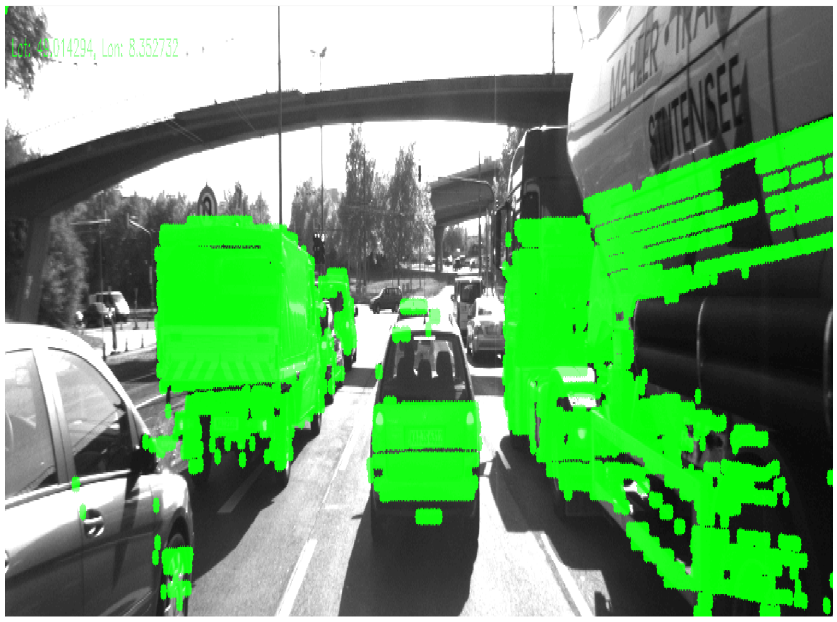

Figure 2: Final Sensor Fusion Output - still frame from video

Code Analysis

This segment of the report is dedicated to a granular analysis of the important functions within the codebase, illuminating their specific roles and contributions to the project's overarching objectives.

transformLidarPoints Kernel

Purpose: Converts 3D LIDAR points to 2D image points for overlay on camera images.
Functionality: This CUDA kernel operates in parallel across multiple threads, efficiently transforming LIDAR data using matrix multiplication and shared memory for enhanced performance. It employs conditional checks to filter out points based on predefined boundaries and reflection intensity, ensuring only relevant data is processed.
Interactions: Receives LIDAR data and calibration matrices as input and outputs transformed 2D points for visualization.

readGPSData Function

Purpose: Reads real-time GPS data from a specified file format.
Functionality: Opens and reads the GPS data file, parsing latitude and longitude values. Implements error handling to manage file access and parsing issues.
Interactions: Supplies latitude and longitude data to the overlayGPS function for visual representation.

overlayGPS Function

Purpose: Overlays GPS data onto the camera images to provide real-time positional information.
Functionality: Utilizes OpenCV functions to create a transparent overlay on the original image, where the GPS data is rendered. This enhances the data presentation by adding contextual location information.
Interactions: Works in conjunction with the image processing workflow to incorporate GPS text onto the final displayed image.

projectLidarToCamera Function

Purpose: Orchestrates the workflow for processing and visualizing LIDAR and camera data, including GPS overlay.
Functionality: Manages the sequence of operations, including reading sensor data, invoking CUDA kernels for data transformation, and integrating GPS data. Utilizes OpenCV for image handling and CUDA for parallel processing of LIDAR data.
Interactions: Central function that ties together data reading, processing, and visualization components.

Load and Conversion Functions

loadCalibrationData, convertMatToFloatArray: These functions handle the loading of calibration data necessary for accurate LIDAR to camera projection and convert OpenCV Mat objects to arrays suitable for CUDA operations.

Challenges and Solutions

CUDA Kernel Development

Developing the CUDA kernel proved to be a complex undertaking, necessitating a deep understanding of parallel computing principles and optimization techniques. The use of NVIDIA's Nsight Compute tool was instrumental in refining the kernel's performance, with the final implementation achieving an impressive execution time of approximately 9 ns per iteration on an RTX 3090 GPU.

OpenCV CUDA Integration

Integrating CUDA-accelerated OpenCV functions presented significant challenges, primarily due to issues with CMake and outdated CUDA support within the OpenCV library. Overcoming these obstacles required extensive troubleshooting and adaptation of the build process to ensure compatibility with the project's requirements.

LIDAR Data Processing

The initial approach to processing LIDAR data involved converting binary files to text, resulting in a substantial bottleneck that significantly impeded the system's performance. By eliminating this conversion step and directly processing binary files, the project was able to alleviate this issue, dramatically reducing data processing times.

Color Gradient Implementation

The inability to implement a color gradient for the LIDAR overlay due to the lack of equivalent CUDA-accelerated functions in OpenCV posed a significant challenge. The project's tight timeline precluded the development of a custom kernel for this purpose, and the reliance on GPU processing for point computations made it impractical to revert to CPU-based operations for color grading.

Discussion

The project's success in transitioning to a CUDA-based sensor fusion system represents a significant leap forward in the field of AV technology. The enhanced processing speed and efficiency afforded by CUDA parallel computing have opened new avenues for real-time data processing and integration. However, the challenges encountered during the project, particularly in terms of CUDA kernel development, OpenCV integration, and the relatively-tame throughput improvements highlight the complexities and potential pitfalls of such an endeavor. The following section will discuss the implications of current CUDA performance.

Analysis of Performance Improvement

The transition to a CUDA-based system for sensor fusion in autonomous vehicles was anticipated to yield significant gains in processing speed and throughput. The project achieved a 2.5x improvement in performance when compared to the CPU-based implementation. While this represents a notable enhancement, it fell short of the expected exponential increase in the range of 30x to 40x. This discrepancy prompts a critical examination of the underlying factors that may have constrained the system's performance.

Factors Contributing to Lower-than-Expected Throughput

Frequent Memory Transfers

One of the primary factors suspected to impact the system's efficiency is the frequent memory transfers between the CPU and GPU. Data transfer over the PCIe bus is a known bottleneck in CUDA applications, as it involves significant latency and bandwidth limitations compared to the GPU's internal processing capabilities. In the context of this project, the sensor data must be transferred from the CPU to the GPU for processing and then back to the CPU for display. These transfers can significantly impede throughput, particularly if the data size is substantial or if transfers are not optimally managed.

Lack of Shared Memory Utilization

The CUDA programming model offers shared memory as a faster, albeit limited, alternative to global memory, enabling threads within the same block to share data efficiently. The underutilization of shared memory in the project's CUDA kernels could lead to suboptimal performance, as global memory accesses are considerably slower and can lead to memory bandwidth saturation. Further leveraging of shared memory for frequently accessed data or for intermediary computations within thread blocks could potentially alleviate some of the performance constraints.

Limited Dataset

The scale of data used for processing also plays a crucial role in realizing the full potential of GPU acceleration. GPUs are most effective when the amount of parallelism justifies the overhead associated with launching kernels and managing GPU resources. In this project, the relatively low number of images processed might not have provided sufficient workload to fully leverage the GPU's parallel processing capabilities, leading to underutilization of the available computational resources.

Strategies for Future Optimization

To address these challenges and improve system performance, several optimization strategies can be considered:

> *Optimizing Data Transfer*
> Employing CUDA streams and asynchronous memory transfers can help overlap data transfer with kernel execution, reducing the impact of PCIe bus latency. Additionally, minimizing the amount of data transferred or using pinned memory can improve transfer rates.
>
> *Maximizing Shared Memory Usage*
> Revising the CUDA kernels to make better use of shared memory for frequently accessed data or for data shared among threads in a block can reduce global memory accesses and improve overall performance.
>
> *Scaling the Dataset*
> Increasing the number of images or the size of the datasets processed can help ensure that the GPU's computational resources are fully utilized, thereby increasing the efficiency and throughput of the system.
>
> *Kernel Optimization*
> Further optimizing the CUDA kernels by examining thread and block configurations, minimizing divergent execution, and reducing warp serialization can also contribute to better performance.

By addressing these factors, there is potential to significantly enhance the throughput of the CUDA-based sensor fusion system, potentially achieving the exponential performance improvements initially anticipated.

Conclusion

This project has demonstrated the viability and benefits of employing CUDA for sensor fusion in autonomous vehicles, achieving modest improvements in processing speed and efficiency. Despite the challenges faced, the project's outcomes provide a valuable foundation for future research and development in this area. As AV technology continues to evolve, the lessons learned and methodologies developed through this project will undoubtedly contribute to the advancement of sensor fusion techniques and the broader field of autonomous navigation.

References

NVIDIA Corporation. (n.d.). CUDA Toolkit Documentation. Retrieved from https://developer.nvidia.com/cuda-toolkit

Tooth, P. (n.d.). 2D Sensor Fusion. GitHub repository. Retrieved from https://github.com/tooth2/2D-Sensor-Fusion

Project code can be found here: https://github.com/SizeDrip/Past-Projects/tree/2e02872dde4c11ea7d61e0e4a1b43fb675cadc31/2D-Sensor-Fusion-main