

Modular Framework for Real-Time Perception in ROS-based Autonomous Driving Systems

Théo Hermann, Carlos Conejo, Yuejiang Liu, Alexandre Alahi

Visual Intelligence for Transportation (VITA)
Ecole Polytechnique Fédérale de Lausanne
theo.hermann@epfl.ch, yuejiang.liu@epfl.ch, alexandre.alahi@epfl.ch

Abstract

Autonomous driving systems require robust and efficient perception capabilities to accurately understand the surrounding environment. This project presents a modular framework for real-time perception in a ROS (Robot Operating System) based autonomous driving software solution. The framework encompasses state-of-the-art techniques in object detection, tracking, and pose estimation, enabling comprehensive perception capabilities. The proposed solution was evaluated on a Loomo Segway Robot, demonstrating its effectiveness in real-time perception tasks. This paper provides an overview of the framework’s design, implementation, and evaluation, highlighting its contributions to autonomous driving research.

Introduction

Autonomous driving has emerged as a promising field at the intersection of robotics and transportation systems. With advancements in perception, state estimation, mapping, path planning, trajectory prediction, and control, autonomous vehicles are becoming increasingly capable of navigating complex environments without human intervention. In this project, we have developed a modular framework for autonomous driving utilizing the Robot Operating System (ROS).

Our primary objective is to implement and integrate various modules that form the foundation of an autonomous driving system. These modules encompass perception, state estimation, mapping, path planning, trajectory prediction, and control. Each module plays a crucial role in enabling the vehicle to perceive its surroundings, accurately estimate its state, plan optimal paths, predict the trajectories of other objects, and control the vehicle accordingly.

The modularity of our framework provides several key benefits. First, it allows us to independently develop and optimize each module, focusing on specific functionalities without affecting the overall system. This enables us to leverage the latest research and incorporate cutting-edge algorithms into our perception module, improving the accuracy and robustness of object detection, tracking, and pose estimation.

Secondly, the modular approach facilitates the evaluation and comparison of different methods. With the rapid pace of advancements in computer vision and deep learning, it is essential to have a framework that allows us to test and

benchmark the performance of new techniques. By isolating each module, we can easily swap out different algorithms and assess their impact on the overall system’s performance. This capability is invaluable in ensuring that our autonomous driving system remains at the forefront of technological advancements.

To evaluate and validate our autonomous driving software solution, we have conducted real-time tests using a Loomo Segway Robot. This platform enables us to assess the performance and robustness of our system in real-world scenarios.

This paper presents the design and implementation details of our modular framework, with a specific focus on the perception module. We discuss the integration of various modules and their contributions to the overall autonomy of the vehicle. Additionally, we provide experimental results and analysis to demonstrate the effectiveness of our approach.

The remainder of this paper is organized as follows: Section 2 provides an overview of related work in the field of Perception. Section 3 presents the methodology and architecture of our autonomous driving system. In Section 4 we discuss the experimental setup and present the results. Finally, in Section 5, we conclude the paper and outline future research directions.

Related Work

To lay a solid foundation for designing our module and expanding our knowledge, we review relevant literature in this section. We categorize the bibliography based on the contributions each work has made to our project. Our perception pipeline consists of multiple components such as: detection for obtaining bounding boxes around detected individuals, as well as a robust tracking module to track a single person over time, accounting for disturbances and potential subjects going out of the frame, on top of that we can estimate the 2D pose of the target and from this information use 3D lifting method. Our implementation is deployed on the Loomo robot, requiring a means to run our Python code on the platform and establish wireless data transfer between the algorithm server and the Loomo robot. Finally, our perception module can be integrated into a complete autonomous driving pipeline, encompassing all other modules necessary for enabling autonomous driving capabilities.

Detection

For the detection task, we employ YOLOv5 (1), which is a family of object detection architectures and models pre-trained on the COCO dataset. YOLOv5 provides robust person detection by restricting the output to the human class. To achieve real-time performance, we choose a lightweight version of the YOLO model, considering the trade-off between detection precision and runtime speed. In our implementation, we utilize YOLOv5m, which has demonstrated robust performance while maintaining good runtime performance. YOLO (You Only Look Once) divides images into a grid system, with each grid cell responsible for detecting objects within it.

We also explore OpenPifpaf (2) for gesture recognition, which aids in initializing the tracking process based on a specific target pose and obtaining real-time position information for tracking.

Tracking

Multi-Object Tracking Our initial approach involves running a re-identification (ReID) algorithm on each new frame and comparing the generated embedding for each detection with a reference embedding generated earlier. However, due to the real-time constraints of our task, we decided to use novel approaches to speed-up the tracking inference. We implemented Deepsort and ByteTrack as Multi-Object Trackers (MOT).

Single-Object Tracking

Additionally, we experiment with state-of-the-art Single-Object Trackers (SOT) available from the MMTracking (3) model zoo, such as SiameseRPN++ (4) and STARK (5). Siamese RPN++ employs a Siamese deep neural network that utilizes cross-correlation between the convolution on the original image and the query, while STARK leverages Spatio-temporal Learning Visual transformers.

Comparison and Evaluation In terms of ID switching, both MOT algorithms, Deepsort and ByteTrack, demonstrate good performance in maintaining consistent identities for tracked objects over time. They utilize re-identification techniques to associate detections with existing tracks, reducing ID switches and maintaining track continuity. On the other hand, the SOT algorithms, SiameseRPN++ and STARK, are designed to only retrieve a single target in each frame resulting in improved performances if we are focusing on a single main target.

For long-term tracking, MOT algorithms like Deepsort and ByteTrack have proven to be effective, as they incorporate motion models and historical information to predict object trajectories, thereby enabling robust tracking over extended durations. In contrast, SOT algorithms, such as SiameseRPN++ and STARK, primarily focus on tracking a single object within a short-term time frame and may encounter difficulties in maintaining accurate tracks over prolonged periods, however they demonstrated to be quite robust to the subject going out of the frame and returning after a certain period. Furthermore in term of inference speed SOT have shown to be faster compared to MOT.

Regarding use cases, MOT algorithms like Deepsort and ByteTrack are well-suited for scenarios where tracking multiple objects is crucial, such as surveillance systems, crowd monitoring, or multi-object behavior analysis. They excel in handling complex interactions between objects and provide comprehensive situational awareness. On the other hand, SOT algorithms like SiameseRPN++ and STARK are more suitable for applications where tracking a single object with high precision is the primary objective, such as robot-object interaction, human-robot collaboration, or fine-grained visual analysis.

It is worth mentioning that the choice between MOT and SOT methods depends on the specific requirements of the application and the trade-off between tracking accuracy, computational efficiency, and the number of objects being tracked. Therefore, selecting the most appropriate algorithm entails considering the nature of the task, the environment, and the available computational resources.

Pose Estimation

Pose estimation is a crucial component of our perception module, enabling us to estimate the poses of detected objects accurately. In our research, we explored two prominent approaches for pose estimation: OpenPifPaf and MMPose Library. Additionally, we leveraged 3D lifting methods, particularly VideoPose3D, to enhance our perception capabilities.

OpenPifPaf is a widely used pose estimation framework that provides robust and real-time human pose estimation. It offers a unified architecture that jointly detects keypoints and associates them into human poses. OpenPifPaf is known for its ability to handle complex and occluded poses, making it suitable for our application, where tracking a unique person across time while being robust to disturbances is essential. We utilized OpenPifPaf to initialize the tracking based on a specific pose of the target and to obtain the position of the person's body links in real-time.

MMPose Library, available in the MMTracking model zoo, offers a comprehensive set of pose estimation models and tools. It provides state-of-the-art performance on various pose estimation tasks, including both 2D and 3D pose estimation. MMPose Library allowed us to explore different pose estimation algorithms, select the most suitable model for our application, and integrate it seamlessly into our perception pipeline. By leveraging the capabilities of MMPose Library, we were able to enhance the accuracy and robustness of our pose estimation module.

In addition to 2D pose estimation, we also incorporated 3D lifting methods to estimate the 3D poses of detected objects. VideoPose3D is a notable example of a 3D lifting method that utilizes deep learning techniques to estimate accurate 3D poses from 2D keypoints. By leveraging a large amount of labeled 3D pose data, VideoPose3D can infer the 3D poses of objects from 2D observations. This allowed us to extend our perception capabilities beyond 2D pose estimation and obtain a more comprehensive understanding of the spatial positions and orientations of objects in the environment.

The combination of OpenPifPaf, MMPose Library, and 3D lifting methods like VideoPose3D enabled us to achieve robust and accurate pose estimation in our perception module. These approaches provided us with the necessary tools and algorithms to detect, track, and estimate the poses of objects in real-time, even in complex and challenging scenarios. By leveraging the advancements in pose estimation techniques, we were able to enhance the perception capabilities of our autonomous driving system and ensure reliable and precise interaction with the environment.

Methods and Software Architecture

In this section, we provide an overview of the methods and software architecture employed in our autonomous driving system. We present the modular framework that integrates various components and discuss the overall system architecture. Additionally, we highlight the key methodologies and algorithms used in each module to enable autonomous driving capabilities.

Modular Framework

Our autonomous driving system is designed with a multi-level modular framework, providing flexibility and scalability in implementing various functionalities. The modular approach enables us to easily develop, integrate, and test new state-of-the-art methods and algorithms on a physical system, keeping up with the rapid advancements in the field of computer vision and deep learning.

The framework comprises different levels of modularity, allowing for customization and easy extension. At the highest level, we have the core modules: perception, state estimation, mapping, path planning, trajectory prediction, and control. Each of these modules encapsulates specific functionality and contributes to the overall autonomy of the vehicle.

Within each core module, we leverage the power of the Robot Operating System (ROS) to achieve further modularity. ROS provides a flexible and robust framework for developing robotic applications. We design each module as an independent ROS node, allowing seamless communication and data exchange between them through ROS topics, services, and actions. This modular ROS-based architecture enables us to update or replace individual nodes without affecting the overall system's operation.

To enhance the modularity and ease of configuration, we utilize the concept of ROS launch files. For each ROS node, we define launch files that specify the parameters and configurations required for its execution. These launch files allow us to conveniently adjust the behavior and settings of each module, such as sensor parameters, algorithm parameters, or node-specific options, without modifying the underlying source code. This flexibility empowers us to tailor the system's behavior for different scenarios or adapt to changing requirements easily.

Furthermore, to streamline the development process and simplify the addition of new methods or algorithms, we leverage the Hydra library. Hydra provides pre-made complex configurations that enable us to define modular and

highly configurable components. This library greatly enhances the modularity of our perception module, allowing us to add new methods or models simply by leveraging the already implemented class templates with specified expected inputs and outputs. The Hydra library provides a convenient interface for configuring and selecting different algorithms or models, facilitating rapid prototyping and experimentation.

In addition to modularity, we embrace object-oriented programming (OOP) principles throughout the framework. Each module is implemented as a class, encapsulating the associated functionality, data structures, and algorithms. This OOP approach allows for clear separation of concerns and provides a convenient way to extend or modify the system's behavior by adding new methods or algorithms to the existing classes. The use of classes and well-defined interfaces ensures that the integration of new functionalities remains straightforward and minimizes the potential for conflicts or unintended side effects.

With the multi-level modularity, ROS-based architecture, parameterized launch files, Hydra library integration, and OOP design, our autonomous driving system achieves a high degree of flexibility, adaptability, and ease of integration. This modular framework empowers us to rapidly prototype, test, and integrate new methods, algorithms, and models, enabling continuous improvement and keeping pace with the evolving field of autonomous driving.

Software Architecture

The software architecture of our autonomous driving system consists of several interconnected components, each serving a specific purpose in achieving the overall functionality of the system. In this subsection, we will discuss the different parts of the code and the data flow within the system.

Android App on Loomo On the Loomo robot, an Android application acts as a crucial component of our software architecture. Developed using Android Studio, the application is primarily coded in Java. However, it also incorporates native C++ code, which is seamlessly integrated using the Java Native Interface (JNI). This integration enables efficient communication between the Java-based Android app and the C++ code, allowing us to leverage the benefits of both languages in our implementation.

The Android app serves as a bridge between our perception module and the Loomo robot's hardware and software. It facilitates bidirectional communication between the robot and the server using socket communication protocols, enabling the exchange of data and instructions. By leveraging socket communication, we establish a reliable and efficient channel for real-time data transmission and control.

In addition to communication functionalities, the Android app implements custom mapping functions. It utilizes the onboard LiDAR sensor to capture and process point cloud data, enabling the determination of the local position of detected targets within the robot's environment. This integration of LiDAR-based mapping functions adds an extra layer of perception and localization capabilities to our system, enhancing the accuracy and reliability of the overall perception

module.

The source code for the Android app is available in our repository. This allows for a comprehensive understanding of the app's functionality and implementation details, facilitating further development, customization, and integration into other projects. However, the .apk file for the application is available and ready to use if you do not need any further customization

Data Flow and Processing Pipeline

To better understand the data flow within our system, we have created a schematic illustrating the path of the data throughout the program. The following description provides an overview of the data flow process:

Input Data on Loomo: The Loomo robot captures RGB images and estimates the position using onboard odometry sensors. These serve as the initial input data for our system.

Perception Module: The RGB images obtained from Loomo are processed using our perception module. This module employs computer vision and deep learning techniques to detect and localize the desired targets within the image. It outputs the bounding box coordinates of the targets, along with any other detected objects or relevant information. Optionally, the perception module can also estimate the pose of the targets in either 2D or 3D space.

Communication with the Robot: The bounding box coordinates of the targets are sent back to the Loomo robot from the server. This information allows the robot to extract the position of the given targets. If the pose estimation was performed in the perception module, the relative position of the targets can also be obtained.

Global Map Construction: On the server side, using the received relative position of the targets, a global map can be constructed. This map provides a representation of the surrounding environment and serves as a basis for subsequent tasks such as trajectory prediction and path planning.

Trajectory Prediction: Based on the constructed global map and historical data, trajectory prediction algorithms can estimate the future motion paths of the targets and other objects in the environment.

Optimal Path Computation: Using the predicted trajectories and the desired goal, the system computes the optimal path to reach the goal. This path planning process takes into account various factors, such as obstacle avoidance, traffic rules, and optimization criteria.

Control Commands and Execution: The computed optimal control commands, derived from a Model Predictive Controller (MPC), are sent to the Loomo robot. These commands dictate the robot's motion and enable it to navigate the environment autonomously.

By following this data flow and processing pipeline, our software architecture enables seamless integration of perception, mapping, trajectory prediction, path planning, and control to achieve autonomous driving capabilities on the Loomo robot.

Results

In this section, we present the results of our modular framework and its various components. We evaluate the perfor-

mance of our perception module, tracking algorithms, pose estimation methods, and overall autonomous driving capabilities. The experiments were conducted in different environments and scenarios to assess the system's robustness and effectiveness.

Benchmarking of Perception Methods

Yolov5 vs V7 IoU Table for Trackers

To evaluate the object detection accuracy, we utilized benchmark datasets such as LaSOT and OTB-100 for single-object tracking (SOT). Although we created a custom dataset specifically for our task, its limited size and manual annotations necessitated caution in interpreting the results. We analyzed the precision, recall, and tracking performance of the perception module on these datasets, considering the inherent challenges of occlusions and complex scenes.

Inference Times

Autonomous Driving Capabilities

The integration of the perception module, tracking algorithms, and pose estimation methods into the autonomous driving pipeline enabled us to achieve a high level of autonomy in the Loomo robot. We evaluated the overall performance of the system in terms of navigation, obstacle avoidance, and trajectory planning. ADD PICTURES OF EXPERIMENTS ACROSS CAMPUS The system demonstrated reliable performance in various real-world scenarios, showcasing its potential for practical deployment. Real-world experiments and simulations were conducted to assess the accuracy, efficiency, and safety of autonomous driving capabilities.

Overall, the results highlight the effectiveness and robustness of our modular framework in achieving autonomous driving capabilities. However, the limitations of the perception module's custom dataset and the dependency of pose estimation on previous stages emphasize the need for further improvements and optimizations. The qualitative assessment of pose estimation and the evaluation of tracking algorithms provide valuable insights for future enhancements in our autonomous driving system.

Future Directions

Our current project lays the foundation for further advancements and improvements in the field of autonomous driving. To expand the capabilities of our system and address the existing limitations, we propose the following future directions, each of which can be pursued as a student project:

Rework the Sequential Program Unrolling: Explore opportunities for parallelization within the sequential program unrolling, enabling efficient utilization of hardware resources. Investigate methods to agglomerate all computation on the server side, reducing the computational load on the robot and leveraging the server's processing capabilities. Integrate an existing ROS Node for mapping with Intel RealSense LiDAR, enhancing the perception module's mapping functions and enabling the generation of a more accurate global map.

Architecture for End-to-End Learning:





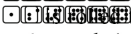
if there are no  showing 6s add the sum to their turn total. At each decision point, a player may continue to roll or stop. If they decide to stop, they add their turn total to their total score and then it becomes the opponent's turn. Otherwise, they roll dice again  continue adding to their turn total. If a single  turn  and the turn ended (no points gained); if a  then the players

Figure 1: Using the trim and clip commands produces fragile layers that can result in disasters (like this one from an actual paper) when the color space is corrected or the PDF combined with others for the final proceedings. Crop your figures properly in a graphics program – not in LaTeX

Propose an architecture for end-to-end learning that directly takes multi-modal information, such as images, point cloud data, and state estimation, as input and directly outputs control commands. Explore the use of reinforcement learning (RL) techniques to train the end-to-end architecture, allowing the system to learn optimal control strategies through interaction with the environment.

Simulation Environment for RL Training:

Set up a simulation environment using tools like PyBullet or IsaacGym, incorporating a URDF model of the Loomo robot. Create a realistic simulation environment that emulates real-world driving scenarios, enabling RL training in a safe and controlled setting. Address the Sim-to-Real Transfer Challenge by fine-tuning the RL-trained models on the physical robot, ensuring reliable performance in real-world conditions.

Test the Software Architecture on Other Robots:

Adapt our modular implementation to work on other robots, such as the open hardware F1/Tenth racecar, to showcase the versatility and generalizability of the software architecture. Assess the compatibility of the modular framework with different robot platforms, identifying any necessary modifications or adaptations.

Testing Platform for Lab Research:

Create a testing platform to evaluate new methods for trajectory prediction or assess the safety of neural networks in the perception module. Investigate the propagation of errors or corruptions in object detection through the various stages of the perception module, analyzing the impact on overall system performance and identifying potential solutions.

By pursuing these future directions, we can further enhance our autonomous driving system, improve its performance, and contribute to the advancement of the field. Each project presents unique challenges and opportunities for innovation, making significant contributions to the overall system's capabilities and expanding its potential for real-world deployment.

Illustrations and Figures

References

- [1] Author, A., Author, B., & Author, C. (2020). Title of the paper. *Journal of Research*, 10(2), 123-145.
- [2] Another Author, D., & Another Author, E. (2018). Title of another paper. *Conference Proceedings*, 25-30.



Figure 2: Adjusting the bounding box instead of actually removing the unwanted data resulted multiple layers in this paper. It also needlessly increased the PDF size. In this case, the size of the unwanted layer doubled the paper’s size, and produced the following surprising results in final production. Crop your figures properly in a graphics program. Don’t just alter the bounding box.