UNVIVERSITY OF CARTHAGE
Higher School of Communication of Tunis

P2M Report

LIDAR-based solution for trajectory prediction

Prepared By :

Fki Edam

Ktari Aymen

Supervised by :

Mrs. ROUISSI Fatma

Mr GRATI Khaled

Academic year :
2023-2024

Contents

List of Figures

Abreviations List

LIDAR : Light Detection And Ranging

GPIO : General Purpose Input/Output

ROS : Robotics Operating System

SLAM : Simultaneous Localisation And Mapping

RNN : Recurrent Neural Network

GRU : Gated Recurrent Unit

LSTM : Long Short-Term Memory

MAE : Mean Absolute Error

MSE : Mean Squared Error

MQTT : Message Queuing Telemetry Transport

General Introduction

Over the years, the integration of LiDAR technology in robotic systems has become crucial across various sectors, playing an essential role in autonomous navigation and trajectory analysis. To meet the growing challenges in this constantly evolving field, it is imperative to incorporate innovative solutions. In this context, the development of a system utilizing LiDAR data within the Robot Operating System (ROS) proves to be a valuable response for accurately predicting the future movements of moving targets.

That is why we have developed a system that integrates advanced data processing techniques and LSTM models to accurately predict the future movements of dynamic targets. By combining LiDAR technology with deep learning, this project aims to provide precise trajectory analysis in the field of robotics.

This innovative approach is particularly useful in applications such as autonomous vehicles, where predicting the movements of pedestrians, cyclists, and other vehicles is crucial for safe navigation. Similarly, in industrial robotics, understanding the movement of objects and co-workers can significantly enhance operational efficiency and safety.

Chapter 1

State of the art and Problematic

Introduction

In this chapter, we will discuss the state of the art and the problematic surrounding trajectory prediction systems for autonomous robot navigation.

1.1 Poblematic

Trajectory prediction systems are crucial for autonomous robot navigation, especially in industrial settings where efficiency and safety are top priorities. These systems allow robots to anticipate their future positions by analyzing past movements, enabling proactive collision avoidance and optimized path planning.

1.2 State of the art

Currently, trajectory prediction technologies primarily utilize four main sensor systems: 3D LiDAR, radar, IMU, and camera-based systems. Below is a table summarizing the characteristics of each system:

Sensor Technology	Description	Strengths	Weaknesses
3D LiDAR	Uses laser pulses for detailed 3D maps.	Accurate 3D mapping, works in various lighting, good for static/dynamic objects.	Limited range, high cost, interference from reflective surfaces.
Radar	Uses radio waves for object detection and velocity.	Works in poor visibility, long-range detection, good for moving objects.	Lower resolution, susceptible to interference.
IMU	Measures changes in acceleration and orientation.	Real-time data, compact, lightweight, complements other sensors.	Drift over time, less accurate for long-term predictions.
Camera	Captures visual information, uses computer vision.	Rich visual data, cost-effective, versatile.	Affected by lighting, limited depth perception, computationally intensive.

Table 1.1 – Main characteristics of trajectory prediction systems

1.3 Proposed solution

Our solution is to implement a trajectory prediction system in autonomous robot . The system employs a LiDAR sensor to continuously scan the environment, creating a detailed 2D map . To facilitate real-time navigation and mapping, SLAM algorithm is used [3], which constructs and updates a map while tracking the robot's position. As the robot moves, its positional data is continuously appended to a trajectory history which is processed and used to train a deep learning model to provide precise and reliable trajectory predictions.

Conclusion

In this chapter, we have discussed the current trajectory prediction systems and introduced our proposed solution for autonomous robot navigation.

Chapter 2

Hardware & Software Environment specifications

Introduction

In this chapter, we will delve into the hardware and software setup crucial for the trajectory prediction system.

2.1 Hardware setup

The trajectory prediction system relies on LiDAR technology for environmental sensing, with the specific LiDAR device utilized in our project being the TG30 ydlidar [1]. This LiDAR unit offers high-resolution scanning capabilities

To facilitate real-time data processing and computation for trajectory prediction, we employed a Raspberry Pi 4 Model B [2], a versatile and cost-effective single-board computer. The Raspberry Pi serves as the central processing unit for the autonomous robot, managing data acquisition and preprocessing.

2.1.1 LIDAR

LIDAR, which stands for Light Detection and Ranging, is a sensor that uses light pulses to measure distance.
It operates by emitting laser pulses and measuring the time it takes for these pulses to reflect off surrounding objects and return to the sensor. By precisely timing the return of these pulses, LiDAR systems can accurately determine the distance to objects in their vicinity.

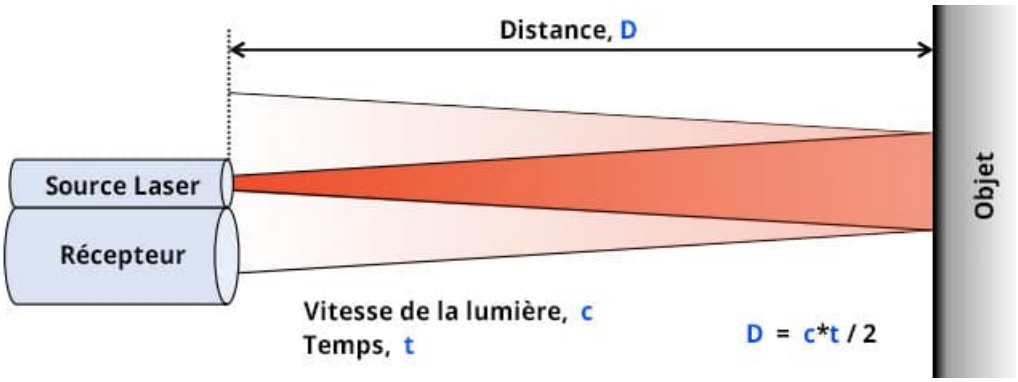

Figure 2.1 – Principle of Operation of a Lidar

2.1.2 Raspberry Pi

The Raspberry Pi is a very cheap computer that runs Linux, but it also provides a set of GPIO pins, allowing you to control electronic components for physical computing and explore the Internet of Things (IoT).

Figure 2.2 – Raspberry Pi Board

2.2 Software environment

In our software environment, we initially utilized the YDLidar software tool on Windows for the initial steps of understanding and visualizing LiDAR data. This tool enabled real-time visualization, noise elimination, scan quality enhancement, and adjustment of scanning parameters. Additionally, the YDLidar software facilitated the storage of collected data from 360° scans.

As we progressed, we transitioned to ROS using a Raspberry Pi operating on Ubuntu 20.04 for system integration and development. With Python, we programmed data reading and processing scripts, enabling efficient utilization of LiDAR data within our trajectory prediction system.

2.2.1 Ydlidar software

The YDLidar software is a tool developed by YDLidar to visualize data collected from their LiDAR sensors in real-time.Users can apply various filters to the data to eliminate noise and enhance the quality of the scans. Additionally, the software allows users to adjust scanning parameters such as the scan frequency. Moreover, the YDLidar software facilitates the storage of the collected data from 360° scans.

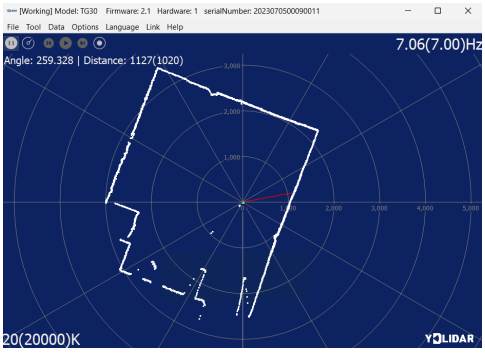

Figure 2.3 – Ydlidar Visualisation TOOL

2.2.2 ROS

The Robot Operating System ROS is a set of open-source software libraries and tools that enhances the scalability and maintainability of robotic systems.ROS provides a rich ecosystem of drivers and developer tools that facilitate seamless integration and robust functionality.

The integration of ROS is crucial for coordinating the Raspberry Pi and LiDAR technology in our project. ROS's standardized communication protocols and hardware abstraction simplify integration, ensuring seamless interoperability. Additionally, ROS's extensive software components accelerate system implementation, enabling us to focus on refining trajectory prediction algorithms. Thus, ROS enhances the scalability and maintainability of our system while efficiently utilizing Raspberry Pi and LiDAR technology.

We have used ROS noetic [4] on ubuntu Focal 20.04.

Figure 2.4 – ROS Verison

Conclusion

Overall, we have established a robust infrastructure combining hardware and software tools to enhance the efficiency of our trajectory prediction system.

Chapter 3

Data manipulation

Introduction

In this chapter, we outline the steps to collect, clean, and prepare data for our trajectory prediction system. Using a robot equipped with a LIDAR sensor, we gathered data, which was then processed and organized for model training and testing.

3.1 Data collection

We used a robot that has a LIDAR sensor mounted on it and connected to a Raspberry Pi board and we began driving the robot remotely to collect its described trajectory.

We made an automated python script that is capable of extracting the trajectory described by the robot in a list format this list is updated continuously to create a dataset as the robot moves the list stores the robots position using slam algorithm to extract robots position

The resulting dataset was then used for training and testing our model. Specifically, the dataset comprised 20,000 sequences. This data collection endeavor spanned approximately 20 hours.

3.2 Data cleaning and preprocessing

We started by sorting through raw LiDAR data, which includes measurements of angles and distances to show what's around. Then, we used SLAM algorithm to clean up this data. SLAM helps us figure out where the LiDAR is moving and where it is in relation to everything else. We made sure to get rid of any measurements where the distance was 0, so our data would be accurate and reliable. We did this carefully to get a dataset showing the continuous path of the LiDAR's movement. Looking at how we cleaned up the data helped us make our prediction system for where the LiDAR will go even better.

3.3 Data Visualisation

Here is a portion of the dataset: a segment of the mobile LIDAR's trajectory composed of its relative coordinates (x,y) :

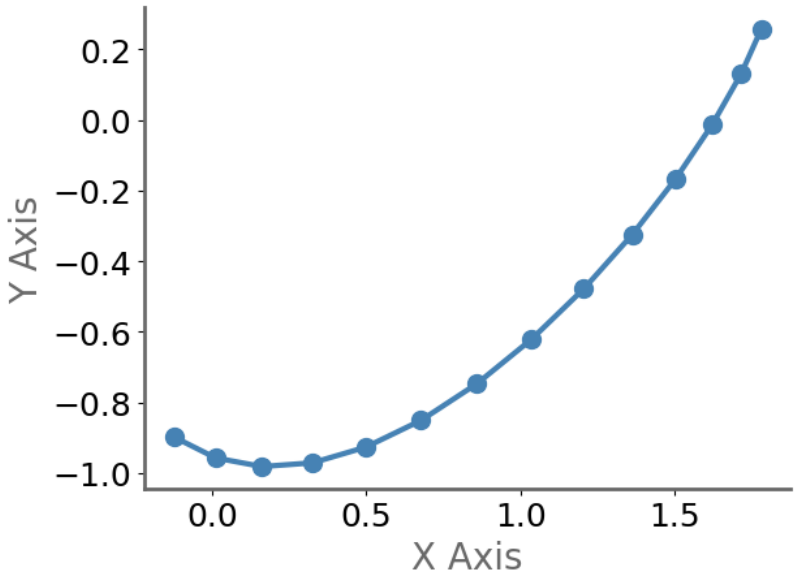

Figure 3.1 – Sample Sequence of the Data

Conclusion

In summary, we successfully collected and processed a comprehensive dataset, ensuring its accuracy and reliability. This prepared dataset is now ready for training and testing our predictive models.

Chapter 4

Model Building

Introduction

In this chapter, we'll focus on model selection, training, performance evaluation, and results.

4.1 Model choice

In selecting the appropriate models for trajectory prediction, we opted for recurrent neural network (RNN)-based architectures, namely GRU, and LSTM variants. These models were chosen due to their inherent ability to effectively handle sequential data and capture temporal dependencies. Unlike traditional feedforward neural networks, which lack memory and fail to consider the sequential nature of time-series data, RNN-based architectures excel in learning patterns from past observations to make accurate predictions about future states, making them ideal in our case .

4.2 Training

To prepare the dataset for model training, we divided it into sequences. Each sequence is composed of 4 relative positions, with the 5th position serving as the label. This structure allows the model to learn from a series of past positions to predict the next one.

Next, we split the dataset into training and testing subsets to evaluate the performance of our models. We allocated 80% of the data for training and 20% for testing to ensure a robust evaluation.

This split resulted in 16,000 sequences for training and 4,000 sequences for testing.

4.3 Model Performance Evaluation

4.3.1 Metrics Choice

To evaluate the performance of our model, we have chosen Mean Squared Error (MSE) as the loss function and Mean Absolute Error (MAE) as the metric for prediction accuracy. MSE is particularly suitable as a loss function because it penalizes larger errors

more significantly, encouraging the model to make precise predictions and minimizing significant deviations from the actual trajectory. On the other hand, MAE is used as an evaluation metric due to its straightforward interpretation as the average magnitude of errors, providing an intuitive measure of prediction accuracy. Together, these metrics ensure a balanced evaluation by focusing on both the overall accuracy and the severity of prediction errors.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}|$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$$

Figure 4.1 – MSE and MAE Formulas

4.3.2 Comparison of Model Performances

We evaluated the performance of each model and plotted the results in the following figure:

Based on this figure, we observed that the LSTM-based model outperformed the GRU-based model in terms of **accuracy and stability** in predicting the LiDAR's next position. Consequently, we chose to use the LSTM-based model in our project, as it demonstrated the best performance for our task.

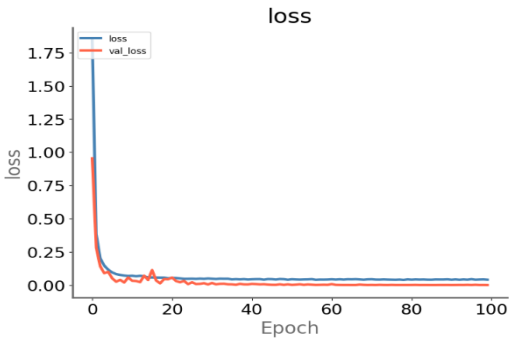

Figure 4.2 – LOSS

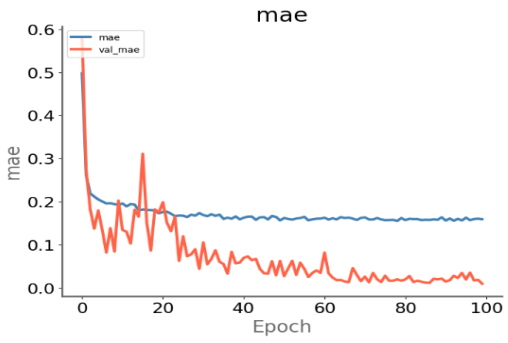

Figure 4.3 – MAE

Figure 4.4 – performances of the LSTM model

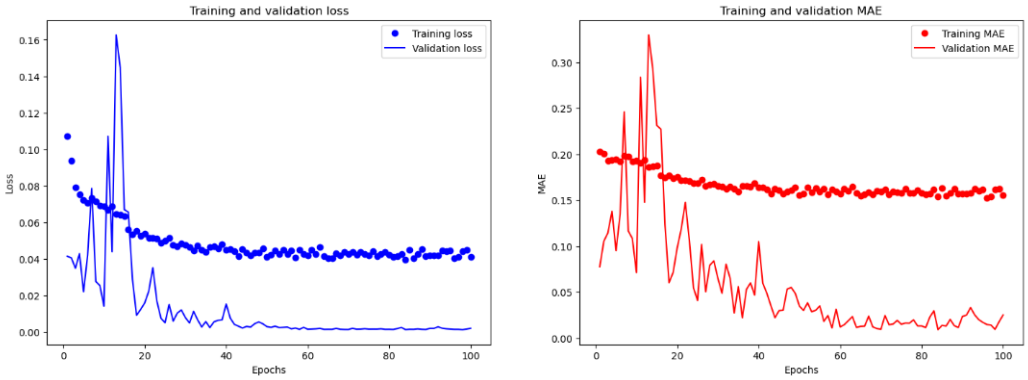

Figure 4.5 – performances of the GRU model

4.4 Results

1-Step Prediction :

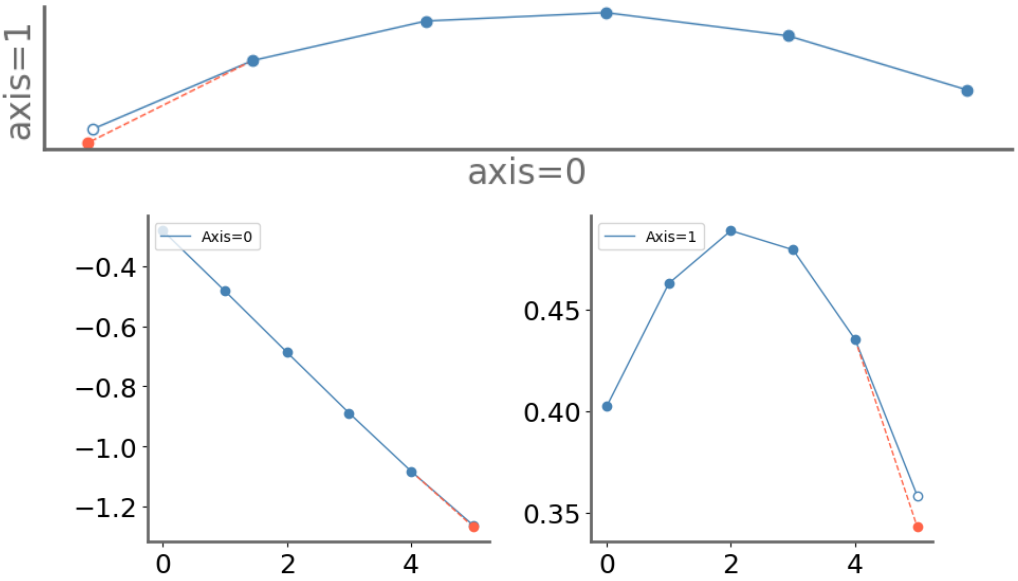

Figure 4.6 – 1-Step Prediction

2-Steps Prediction :

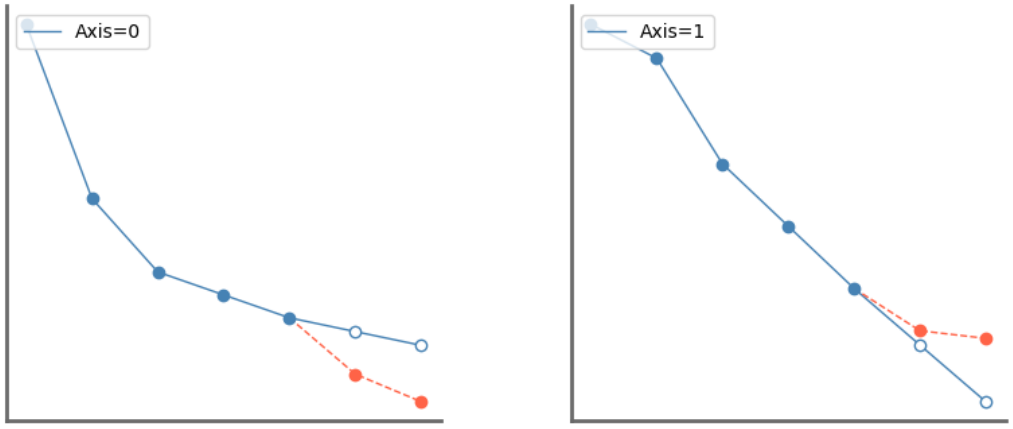

Figure 4.7 – 2-Steps Prediction

Conclusion

In this chapter, we meticulously constructed, trained, and evaluated various RNN models for trajectory prediction, including GRU and LSTM.Our in-depth analysis led us to conclude that LSTM is the most suitable choice for our project.

Chapter 5

Deployment

To deploy our solution, we've installed Ubuntu on a Raspberry Pi 4, followed by the installation of ROS and the necessary LiDAR driver within the Ubuntu environment. Afterward, we mounted this setup onto a remote-controlled robot.Recognizing the Raspberry Pi's limitations in terms of running complex models, we've devised a strategy where the transformed data, specifically the relative coordinates derived from the LIDAR data using SLAM algorithms, are sent over MQTT to our laptop. On the laptop, we've implemented the model execution logic, leveraging its higher computational capacity. Here, the model processes the received data and generates insights such as the trajectory of the robot and forecasts future steps.

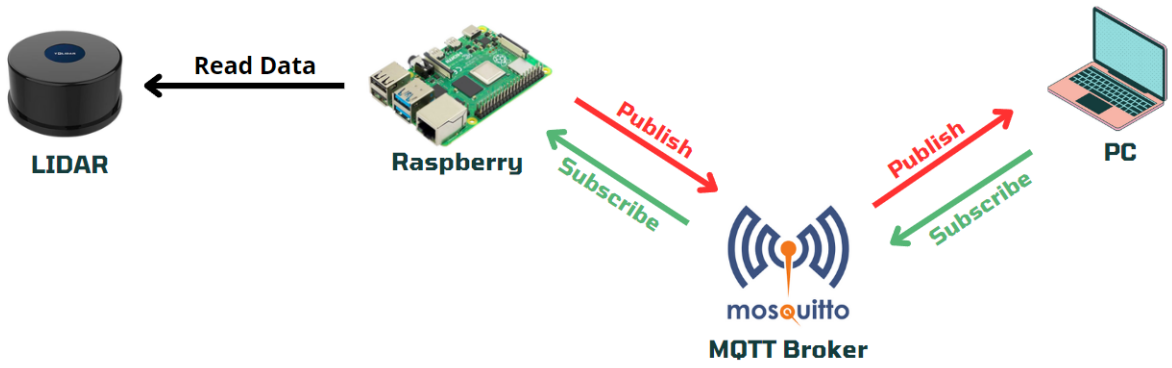

Figure 5.1 – LiDAR Data Communication Flow

Figure 5.2 – Robot

Conclusion

In conclusion, our project has successfully integrated hardware and software components to develop a robust solution for trajectory prediction using LiDAR data. We began by configuring the hardware setup, utilizing a LIDAR sensor connected to a Raspberry Pi 4, providing cost-effective computing capabilities along with GPIO pins for physical computing. Leveraging the versatility of the Robot Operating System (ROS) and the YDLidar software, we enhanced data visualization and manipulation capabilities, enabling real-time data processing and storage. Our data collection and preprocessing methods, including SLAM algorithms, facilitated the construction of a comprehensive dataset containing the trajectory of the moving LiDAR.

Moving forward, our model building phase involved selecting and training RNN-based models such as Simple RNN, GRU, and LSTM to predict trajectory with high accuracy. Through rigorous evaluation and comparison, we identified the LSTM-based model as the most suitable choice due to its superior performance in capturing temporal dependencies. We further evaluated model performance using metrics such as Mean Squared Error (MSE) and Mean Absolute Error (MAE), ensuring a balanced assessment of prediction accuracy.

Finally, for deployment, we optimized resource utilization by offloading model execution to a laptop with higher computational capacity, while the Raspberry Pi focused on data acquisition and communication. By transmitting transformed data over MQTT, our solution enables real-time trajectory prediction and visualization, empowering efficient navigation and decision-making for the remote-controlled robot. Overall, our project showcases the seamless integration of hardware and software components to address real-world challenges in robotics and IoT applications.

Webography

[1] https://www.ydlidar.com/products/view/14.html

[2] https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

[3] https://people.eecs.berkeley.edu/~pabbeel/cs287-fa09/readings/Durrant-Whyte_Bailey_SLAM-tutorial-I.pdf

[4] https://wiki.ros.org/noetic

[5] https://github.com/tu-darmstadt-ros-pkg/hector_slam

[6] https://gricad-gitlab.univ-grenoble-alpes.fr/talks/fidle/-/tree/master/RNN.Keras3