Ravensburg Weingarten Univeristy of Applied Sceince

Department of Embedded Computing

Robot Speed with CNY70 Sensor

Author:
Anand Kishore Babu Shiji
32277

Supervisor:
Prof. Markus Pfeil

September 3, 2023

Abstract

This project delved into the exploration of optical sensor technology as a means to measure the speed of a robotic system. Focused on the application of CNY70 sensors, the investigation unveiled a crucial revelation: these optical sensors primarily respond to alterations in the distance between the sensor and the ground surface, rather than changes in the robot's velocity. Despite this fundamental challenge, an attempt was made to construct a predictive model using the gathered data. Regrettably, the model's performance was compromised by the inherent inconsistency in the input parameters arising from the sensor's behavior. The project's ultimate conclusion underscores the significance of selecting appropriate sensors aligned with intended objectives and comprehending sensor behavior for successful integration into robotics applications. It firmly establishes that, within the scope of the current hardware configuration, measuring the robot's speed using CNY70 sensors remains unattainable. The findings emphasize the necessity of alternative sensor technologies and methodologies to achieve the desired speed measurement capabilities in future endeavors.

Acknowledgements

I would like to express my profound gratitude to Professor Mr. Pfeil for providing the invaluable opportunity to undertake this project, offering guidance, and sharing expertise throughout its course. I also extend heartfelt thanks to my friends and family for their unwavering support, encouragement, and understanding, which were instrumental in the successful completion of this endeavor. Additionally, I wish to acknowledge the divine guidance and blessings that have illuminated my path throughout this journey.

Contents

List of Figures

Chapter 1

Introduction

Hello [**?**]

1.1 Objectives

The objective of this project is to develop a system that accurately measures the speed of a robot car using CNY70 light sensors placed underneath the car. By analyzing the varying sensor outputs corresponding to different speeds, this project aims to create a dataset that will be used to train a neural network. The trained neural network will serve as a predictive model capable of estimating the car's speed solely based on the real-time sensor readings.

1.1.1 Major Tasks

1. Sensor Integration: Integrate CNY70 light sensors into the robot car's design to capture variations in reflectivity as the car moves across different surfaces.
2. Data Collection: Implement a data collection mechanism that records sensor outputs at different speeds, forming a dataset that pairs sensor readings with corresponding car speeds.
3. Data Preprocessing: Process and clean the collected data, ensuring it is suitable for training the neural network.
4. Neural Network Design: Develop and train a neural network using the preprocessed data to establish a relationship between sensor values and car speeds.
5. Model Validation: Evaluate the accuracy of the trained neural network by validating its predictions against real-world car speeds.
6. Real-time Prediction: Implement the trained model within the robot car to enable real-time predictions of speed based on current sensor readings.

1.1.2 Expected Outputs

1. Accurate Speed Measurement: Successful integration of CNY70 light sensors will enable precise speed measurement based on sensor outputs.
2. Neural Network Model: Creation of a neural network capable of interpreting sensor data to predict the speed of the robot car.
3. Real-time Predictions: Implementation of the trained neural network into the robot car, enabling on-the-fly speed predictions during operation.
4. Insights for Future Applications: The project will provide insights into the relationship between sensor readings and speed, potentially contributing to advancements in autonomous vehicle systems and robotics.

   By accomplishing these objectives, this project will contribute to the advancement of speed measurement techniques for robot cars and pave the way for innovative applications in artificial intelligence and autonomous systems.

1.2 Tools and Software used

1.2.1 Raspberry Pi

I am using the Raspberry Pi 4 for this project, which is already securely mounted on the robot I have received. The Raspberry Pi 4, renowned for its exceptional computing capabilities and versatility, serves as the brains of this robotic system, enabling us to achieve precise speed measurement and real-time data processing.

The Raspberry Pi 4, the latest iteration of the Raspberry Pi single-board computer series, boasts impressive specifications, including a quad-core ARM Cortex-A72 processor, up to 8GB of RAM, multiple USB ports, and Gigabit Ethernet, making it an ideal choice for robotics and IoT applications.

In this project, the Raspberry Pi 4 orchestrates the integration of CNY70 light sensors into the robot's framework, capturing essential data that will be used for speed measurement. Its GPIO (General Purpose Input/Output) pins facilitate seamless sensor interfacing, allowing us to monitor and analyze the varying sensor outputs as the robot navigates different surfaces and speeds.

Furthermore, the Raspberry Pi 4 serves as the hub for data collection, preprocessing, and neural network training. Its computing prowess empowers us to process vast datasets efficiently, ultimately enabling the creation of a robust predictive model. This model, once trained, will provide us with the ability to estimate the robot's speed accurately based on real-time sensor readings.

With its compact form factor, robust processing capabilities, and a vast community of developers and enthusiasts, the Raspberry Pi 4 is an indispensable component of this project. Its utilization not only facilitates our current goals but also opens doors to future enhancements and applications in the realm of robotics and artificial intelligence.

1.2.2 Python

I am utilizing Python as the primary programming language for this project. Python's simplicity and extensive libraries make it an ideal choice for controlling the robot, handling data exchange with the robot, and training the neural network.

Python enables us to write custom scripts for robot control, using the Raspberry Pi's GPIO pins to command motors and sensors for precise movements and data collection.

Additionally, Python's data processing capabilities, with libraries like NumPy and Pandas, are instrumental in gathering and preparing data from CNY70 light sensors. It forms the foundation for training a neural network using machine learning librarie. Here i am using Tensorflow to train my nueral network.

In summary, Python's versatility streamlines the integration of robot control, data processing, and machine learning, creating a unified system for this project and laying the groundwork for future robotics and AI advancements.

1.2.3 ADC

The ADS1015 is a 12-bit analog-to-digital converter (ADC) renowned for its high resolution and versatility. With four input channels and an adjustable gain amplifier, it efficiently converts analog signals into digital data. Its compact size, low power consumption, and I2C interface make it an ideal choice for a wide range of applications, including robotics, environmental monitoring, and industrial automation, providing precise and reliable analog-to-digital conversion.

The ADS1015 is a 12-bit analog-to-digital converter (ADC) of paramount importance in this project. Its significance arises from the fact that the Raspberry Pi, the project's core controller, lacks an onboard ADC. Since the output from the CNY70 light sensors is in analog form, the ADS1015 steps in as the intermediary, meticulously converting these analog signals into digital data. With four input channels and an adjustable gain amplifier, it efficiently fulfills this role, enabling precise and reliable analog-to-digital conversion. Its compact size, low power consumption, and I2C interface make it an ideal choice for a wide range of applications, including robotics, environmental monitoring, and industrial automation.

1.2.4 Tensorflow

TensorFlow is a versatile open-source machine learning framework, and it's a cornerstone of this project. It's used to design and train the neural network responsible for predicting the robot's speed based on data from the ADS1015 ADC. TensorFlow empowers precise model creation and effective training, ensuring the robot can respond accurately to real-time sensor readings. Its wide-ranging capabilities, including neural network architecture design, optimization tools, GPU acceleration, and TensorBoard visualization, make it a vital tool for enhancing the robot's autonomy and performance.

1.2.5 Jupyter notebbok

Jupyter Notebook: For the design and training of the TensorFlow neural network in this project, I harnessed the power of Jupyter Notebook. This interactive web application proved invaluable, allowing me to seamlessly blend code execution, explanations, and visualizations in a single document. With Jupyter Notebook, I could iteratively experiment with diverse network architectures, hyperparameters, and data preprocessing techniques. The code cells provided a dynamic environment for executing Python code step by step, while markdown cells allowed me to document the project's progression, elucidate methodologies, and showcase training outcomes. This collaborative interface greatly enhanced the organization and comprehensibility of the project, enabling efficient development and fine-tuning of the neural network to achieve precise speed predictions.

Chapter 2

Background

2.1 ADC Connection

Here in this project, my idea was to measure the ADC values from the CNY70 sensor at different PWM duty cycles because the speed of the robot car is controlled using PWM signals.

For that, I have created a circuit that converts the analogue signal from the sensor to a digital signal. Here, I am attaching the circuit diagram for the same.

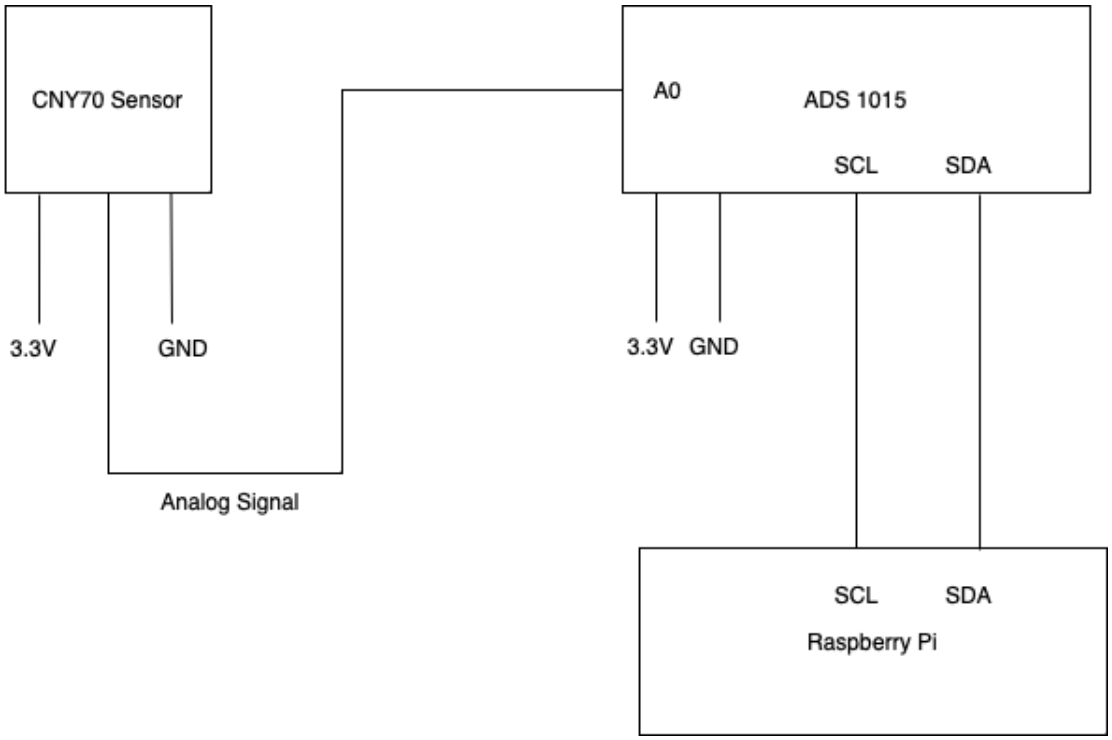

Figure 2.1: Circuit diagram for ADS1015 connection to Raspberry Pi

2.2 Raspberry Pi Pin assignment

Then, different Duty cycles have been tried, and the corresponding ADC values are measured. The measured ADC and Duty cycle values are stored for neural network training. I am attaching the pin allocation diagram for the robot, excluding the ADC. Source :Moodle, Embedded Computing

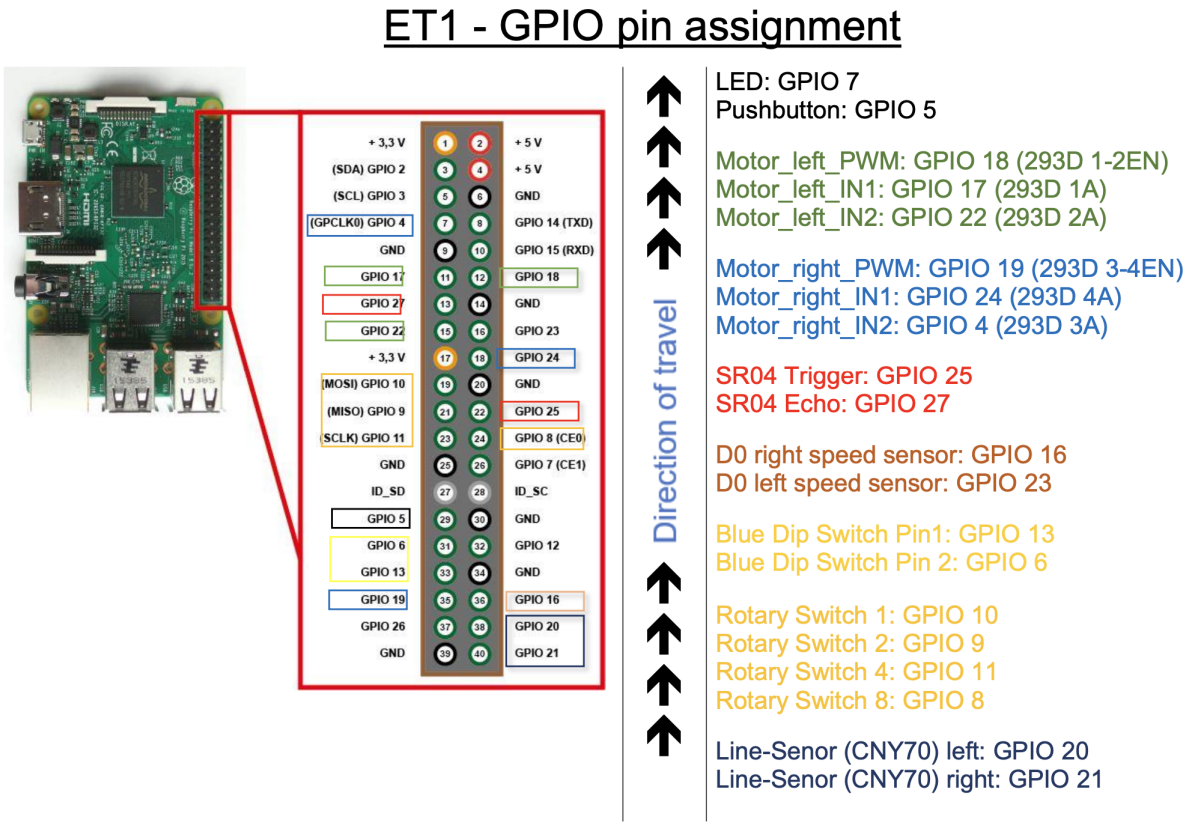

Figure 2.2: GPIO Pin assignment on the Raspberry Pi

2.3 Hardware overview

Here i am adding the diagram of the overview of the hardware. I have changed the direct connection from CNY70 sensors to Raspberry Pi over a ADC which I have described earlier. Also I have used python scripts to read the ADC values at different speeds of robot. Source :Moodle, Embedded Computing

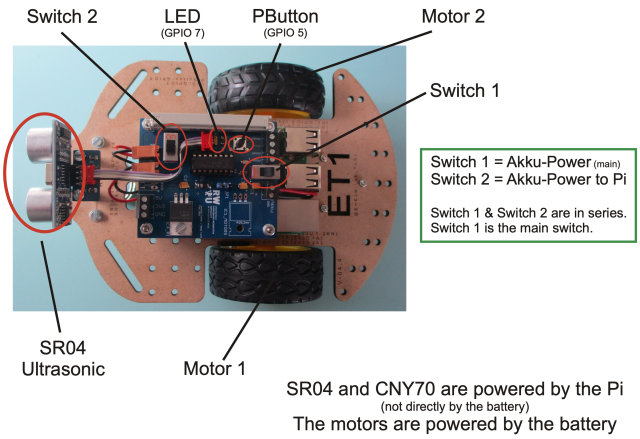

Figure 2.3: Hardware connection of robot

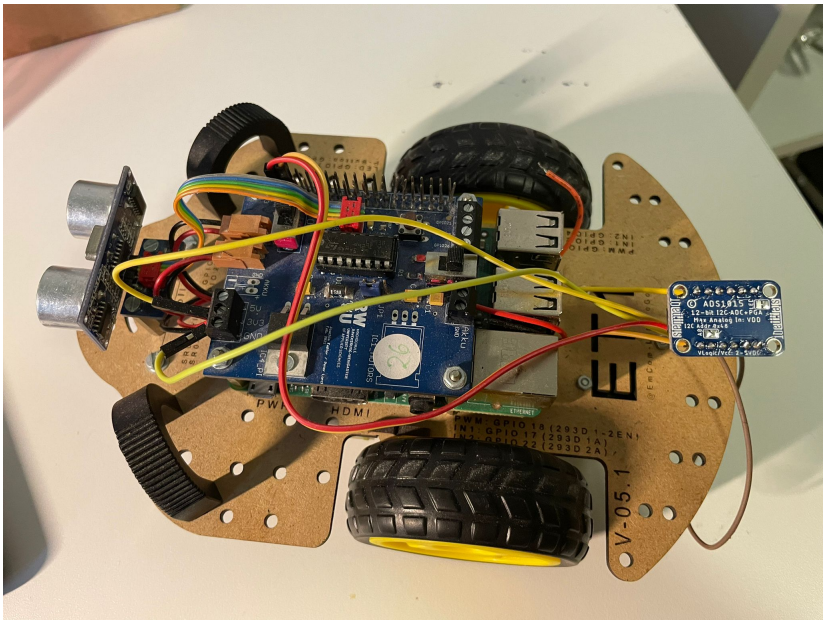

Figure 2.4: Hardware connection with ADC installed

2.4 Neural network details

To predict the speed from the sensor values, I have created a neural network trained using sensor values and the corresponding PWM duty cycles. The previously measured data is given to this network and trained. The architecture and input-output are shown below.

```
In [15]: #input values
sensor_values = [
    60768, 60784, 60784, 60768, 60768, 60736, 60784, 60800, 60784, 60768,
    60768, 60752, 60768, 60768, 60768, 60736, 60768, 60768, 60752, 60784,
    60784, 60784, 60752, 60768, 60784, 60784, 60784, 60736, 60784, 60768,
    60752, 60784, 60832, 60768, 60752, 60784, 60784, 60768, 60752, 60736,
    60800, 60784, 60832, 60784, 60752, 60816, 60784, 60752, 60784, 60832,
    60752, 60816, 60784, 60752, 60784, 60784, 60768, 60768, 60832, 60784,
    60848, 60848, 60848, 60768, 60784, 60848, 60784, 60800, 60784, 60784,
    60784, 60848, 60848, 60768, 60768, 60800, 60784, 60800, 60848, 60784,
    60816, 60832, 60784, 60848, 60768, 60800, 60816, 60832, 60832, 60848,
    60768, 60752, 60784, 60800, 60800, 60768, 60816, 60768, 60816, 60768,
    60752, 60752, 60768, 60768, 60752, 60784, 60784, 60768, 60752, 60768,
    60816, 60832, 60784, 60768, 60784, 60784, 60736, 60752, 60752, 60784,
    60768, 60784, 60784, 60768, 60768, 60736, 60784, 60800, 60784, 60768,
    60800, 60784, 60832, 60784, 60752, 60816, 60784, 60752, 60784, 60832,
    60752, 60752, 60768, 60768, 60752, 60784, 60784, 60768, 60752, 60768,
]
```

```
In [16]: #output values
duty_cycle = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
              0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
              0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
              30, 30, 30, 30, 30, 30, 30, 30, 30, 30,
              30, 30, 30, 30, 30, 30, 30, 30, 30, 30,
              30, 30, 30, 30, 30, 30, 30, 30, 30, 30,
              60, 60, 60, 60, 60, 60, 60, 60, 60, 60,
              60, 60, 60, 60, 60, 60, 60, 60, 60, 60,
```

Figure 2.5: Input and Output dataset for Neural network

Model details

The neural network model is designed for a specific task of predicting the duty cycle based on a single input feature, which represents sensor values. The model is structured into three layers. The first layer consists of 64 neurons, and each neuron in this layer processes the input data using the Rectified Linear Unit (ReLU) activation function. This helps capture complex patterns in the sensor data.

The second layer follows with 32 neurons, also utilizing the ReLU activation function. These layers are designed to extract and represent relevant features from the sensor values, allowing the model to learn and understand the relationships between sensor data and duty cycle.

The final layer, named duty cycle prediction,' comprises a single neuron with a linear activation function. The linear activation function is ideal for regression tasks like duty cycle prediction, as it allows the model to predict continuous numerical values. In this case, it predicts the duty cycle percentage.

In summary, this neural network takes sensor values as input and, through its complex network of neurons and activation functions, learns to make predictions about the duty cycle. It contains a total of 2,241 trainable parameters, which are the internal settings adjusted during training to optimize its predictive accuracy for this specific task.

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                128

 dense_1 (Dense)             (None, 32)                2080

 duty_cycle_prediction (Dens  (None, 1)                33
 e)

=================================================================
Total params: 2,241
Trainable params: 2,241
Non-trainable params: 0
_________________________________________________________________
```

Figure 2.6: Model Achitecture

Training Graph

This is the training loss graph of the model I previously discussed. The training loss graph visually represents the performance of our neural network during the training process. It tracks two key metrics: training loss and validation loss. The training loss, which measures the error between our model's predictions and the actual training data, is expected to decrease steadily during training, indicating that the model is learning and improving its fit to the training dataset. Simultaneously, we also monitor the validation loss, which gauges how well our model generalizes to unseen data. Ideally, the validation loss should follow a similar decreasing trend, signifying that the model is not only learning from the training data but also maintaining its ability to make accurate predictions on new, unseen data. This loss graph is a vital tool in assessing the model's performance, identifying potential overfitting, and guiding decisions on the model's training duration and any necessary adjustments to improve its overall effectiveness.

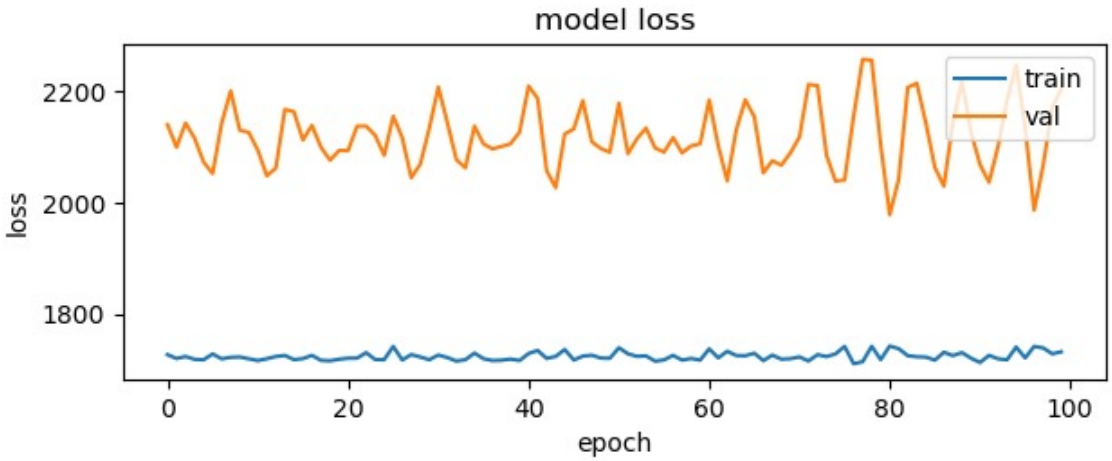

Figure 2.7: Training graph

Chapter 3

Results

3.1 Readings from Sensor

Here, I am adding the readings from the CNY sensor at different PWM duty cycles. The measurement is taken while running the ET1 robot at different speeds. The analog signals from the sensor are converted with the ADS1015, and the Raspberry Pi measures the digital signals coming from the ADC.

3.1.1 At 0% Duty Cycle (Robot at stand still)

Sensor value: 60768
Sensor value: 60784
Sensor value: 60784
Sensor value: 60768
Sensor value: 60768
Sensor value: 60736
Sensor value: 60784
Sensor value: 60800
Sensor value: 60784
Sensor value: 60768
Sensor value: 60768
Sensor value: 60752
Sensor value: 60768
Sensor value: 60768
Sensor value: 60768
Sensor value: 60736
Sensor value: 60768
Sensor value: 60768
Sensor value: 60752
Sensor value: 60784
Sensor value: 60784
Sensor value: 60784
Sensor value: 60752
Sensor value: 60768

3.1.2 At 30% Duty Cycle (Robot moving slowly)

Sensor value: 60752
Sensor value: 60784
Sensor value: 60832
Sensor value: 60768
Sensor value: 60752
Sensor value: 60784

Sensor value: 60784
Sensor value: 60768
Sensor value: 60752
Sensor value: 60736
Sensor value: 60800
Sensor value: 60784
Sensor value: 60832
Sensor value: 60784
Sensor value: 60752
Sensor value: 60816
Sensor value: 60784
Sensor value: 60752
Sensor value: 60784
Sensor value: 60832
Sensor value: 60752
Sensor value: 60816
Sensor value: 60784
Sensor value: 60752
Sensor value: 60784

3.1.3 At 60% Duty Cycle

Sensor value: 60848
Sensor value: 60848
Sensor value: 60848
Sensor value: 60768
Sensor value: 60784
Sensor value: 60848
Sensor value: 60784
Sensor value: 60800
Sensor value: 60784
Sensor value: 60784
Sensor value: 60784
Sensor value: 60848
Sensor value: 60848
Sensor value: 60768
Sensor value: 60768
Sensor value: 60800
Sensor value: 60784
Sensor value: 60800
Sensor value: 60848
Sensor value: 60784
Sensor value: 60816
Sensor value: 60832
Sensor value: 60784
Sensor value: 60848
Sensor value: 60768
Sensor value: 60800

3.1.4 At 90% Duty Cycle (Robot moves fast

Sensor value: 60768
Sensor value: 60752
Sensor value: 60784
Sensor value: 60800
Sensor value: 60800
Sensor value: 60768

Sensor value: 60816
Sensor value: 60768
Sensor value: 60816
Sensor value: 60768
Sensor value: 60752
Sensor value: 60752
Sensor value: 60768
Sensor value: 60768
Sensor value: 60752
Sensor value: 60784
Sensor value: 60784
Sensor value: 60768
Sensor value: 60752
Sensor value: 60768
Sensor value: 60816
Sensor value: 60832
Sensor value: 60784
Sensor value: 60768

3.2 Model Training results

I have created and trained a neural network from the above-obtained values to predict the duty cycle from the sensor values obtained. Here, I am adding the training graph of the model.

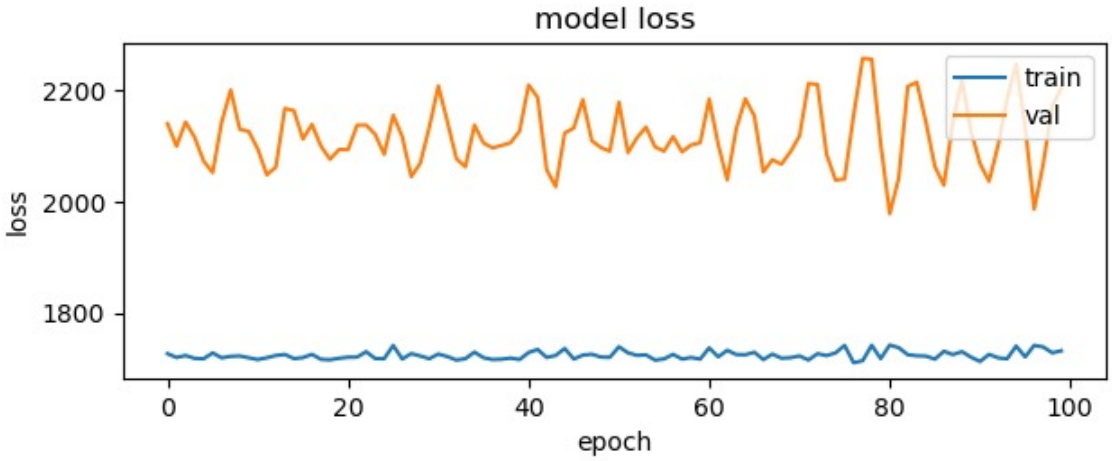

Figure 3.1: Training graph

Chapter 4

Conclusion

In conclusion, this project aimed to explore the feasibility of utilizing CNY70 sensors for measuring the speed of a robotic system. Through meticulous experimentation and data collection, it has become evident that the CNY70 sensor's output remains relatively consistent across a range of robot speeds. Notably, the sensor's response primarily varies in response to alterations in the distance between the sensor and the ground surface rather than fluctuations in the robot's velocity.

Despite the inherent challenges posed by this sensor behaviour, an attempt was made to develop a predictive model utilizing the collected data. Regrettably, the model exhibited significant limitations, yielding high loss values and poor accuracy. This underperformance can be attributed to the inherent inconsistency in the input parameters stemming from the sensor's behaviour.

In light of these findings, it can be concluded that, given the current setup and hardware configuration, measuring the robot's speed using CNY70 sensors appears unfeasible. The project's outcomes underscore the importance of selecting appropriate sensors and methodologies aligned with the intended objectives and the necessity of comprehensively understanding sensor behaviour for successful integration into robotics applications. Future endeavours in this domain require alternative sensor technologies and approaches to achieve the desired speed measurement capabilities.

4.1 Working of Optical Mouse

At its core, an optical mouse incorporates a specialized optical sensor, often in the form of an LED (Light Emitting Diode) or a laser diode. This sensor serves as the mouse's "eye" and is responsible for gathering essential data for tracking movement. When the optical mouse is in use, the sensor emits a focused beam of light, typically red or, in the case of laser-based sensors, invisible infrared light, onto the surface below it.

The magic happens when this emitted light interacts with the surface. Upon striking the surface, the light beam undergoes various optical interactions, including scattering and reflection. The specifics of these interactions depend on the characteristics of the surface itself, such as its texture, color, and reflectivity.

The optical sensor, acting as a light detector, continuously captures and records these intricate light patterns as the mouse is moved. These patterns, which can change rapidly, are then subjected to a complex process of image processing. Inside the mouse, a dedicated sensor processor or chip processes the collected data, and this is where the real magic unfolds.

The heart of the image processing involves pattern recognition algorithms. These algorithms scrutinize the changes in the captured light patterns over time. By comparing consecutive images or measurements, the processor can discern how the patterns have evolved. In doing so, it determines critical information about the mouse's movement, including its direction, speed, and distance covered.

This wealth of movement data, now accurately quantified, is then translated into precise cursor movements on the computer screen. The computer's operating system interprets the information transmitted by the mouse and adjusts the position of the on-screen cursor accordingly. This entire process occurs in real-time, resulting in the seamless and responsive cursor control that we associate with optical mice.

One of the significant advantages of optical mice over their mechanical counterparts is their reliability. They lack the moving parts, like rolling balls, which can become dirty, obstructed, or wear out over time. As such, optical mice have become the standard for modern computer input devices, offering users a more precise and low-maintenance solution for navigating digital interfaces [1].

4.2 Code snippets

```
import time
import smbus
import RPi.GPIO as GPIO


GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

duty_cycle = 0

LED = 7

Motor1_PWM = 18
Motor1_IN1 = 17
Motor1_IN2 = 22

Motor2_PWM = 19
Motor2_IN1 = 24
Motor2_IN2 = 4

def M1_forward():
    GPIO.output(Motor1_IN2,GPIO.LOW)
    GPIO.output(Motor1_IN1,GPIO.HIGH)

def M1_backward():
    GPIO.output(Motor1_IN1,GPIO.LOW)
    GPIO.output(Motor1_IN2,GPIO.HIGH)

def M2_forward():
    GPIO.output(Motor2_IN2,GPIO.LOW)
    GPIO.output(Motor2_IN1,GPIO.HIGH)

def M2_backward():
    GPIO.output(Motor2_IN1,GPIO.LOW)
    GPIO.output(Motor2_IN2,GPIO.HIGH)


GPIO.setup(Motor1_IN1,GPIO.OUT)
GPIO.setup(Motor1_IN2,GPIO.OUT)
GPIO.setup(Motor1_PWM,GPIO.OUT)
PWM_1 = GPIO.PWM(Motor1_PWM, 90) #GPIO als PWM mit Frequenz 90Hz
PWM_1.start(0) #Duty Cycle = 0

GPIO.setup(Motor2_IN1,GPIO.OUT)
GPIO.setup(Motor2_IN2,GPIO.OUT)
GPIO.setup(Motor2_PWM,GPIO.OUT)
PWM_2 = GPIO.PWM(Motor2_PWM, 90) #GPIO als PWM mit Frequenz 90Hz
PWM_2.start(0) #Duty Cycle = 0
```

```
# Create a new instance of SMBus
bus = smbus.SMBus(1)  # Use 0 for older Raspberry Pi boards

# Address of the ADS1015 ADC (You may need to adjust this if using a different address)
ADS1015_ADDRESS = 0x48

# Configuration for the ADC
ADS1015_CONFIG_REG = 0x01
ADS1015_CONVERSION_REG = 0x00

# Single-shot mode, +/-2.048V range, 128 samples per second
ADS1015_CONFIG_VALUE = 0x8483

# Read the ADC value from a specific channel
def read_adc(channel):
    config = ADS1015_CONFIG_VALUE | (channel << 12)
    bus.write_i2c_block_data(ADS1015_ADDRESS, ADS1015_CONFIG_REG, [(config >> 8) & 0xFF, config &
    time.sleep(0.001)  # Wait for conversion to complete
    data = bus.read_i2c_block_data(ADS1015_ADDRESS, ADS1015_CONVERSION_REG, 2)
    value = (data[0] << 8) + data[1]
    return value
PWM_1.ChangeDutyCycle(90)
M1_forward()

PWM_2.ChangeDutyCycle(90)
M2_forward()


try:
    while True:
        sensor_value = read_adc(0)  # Read from channel 0 of ADS1015
        print("Sensor value: {}".format(sensor_value))
        time.sleep(0.1)
except KeyboardInterrupt:
    pass
```

Chapter 5

Bibliography

1. Optical Mouse." Wikipedia, 26 Aug. 2020, en.wikipedia.org/wiki/Optical$_m ouse$. Optical Mouse." Wikipedia, 26 Aug. 2020, en.wikipedia.org/wiki/Optical$_m ouse$.