UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Speed prediction of a pipeline inspection gauge (PIG) based on differential pressure and acceleration with artificial neural networks

Victor Carvalho Galvão de Freitas

Advisor: Prof. Andres Ortiz Salazar, PhD

Doctoral Thesis presented to the Graduate Program in Electrical and Computer Engineering of UFRN (concentration area: Automation and Systems) in partial fulfillment of the requirements for obtaining the title of Doctor of Science.

PPgEEC Order Number: DXXX
Natal, RN, July 2022

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Central Zila Mamede

Freitas, Victor Carvalho Galvão de.
Speed prediction of a pipeline inspection gauge (PIG) based on differential pressure and acceleration with artificial neural networks / Victor Carvalho Galvão de Freitas. - 2022.
87 f.: il.

Tese (doutorado) - Universidade Federal do Rio Grande do Norte, Centro de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica e de Computação, Natal, RN, 2022.
Orientador: Prof. Dr. Andres Ortiz Salazar.

1. Pipeline Inspection Gauge (PIG) - Thesis. 2. Artificial neural networks - Thesis. 3. Embedded systems - Thesis. 4. Raspberry Pi - Thesis. I. Salazar, Andres Ortiz. II. Título.

RN/UF/BCZM CDU 669-1

Elaborado por Ana Cristina Cavalcanti Tinôco - CRB-15/262

Aos meus queridos pais, José Galvão de Freitas e Alzira Carvalho Galvão de Freitas, por todo o esforço que fizeram para me propiciar a educação que eles jamais puderam obter.

Acknowledgment

To the public servants of the Federal University of Rio Grande do Norte, especially the professors of the Graduate Program in Electrical and Computer Engineering (PPGEEC), for their commitment to providing quality public education.

To the Federal Institute of Education, Science, and Technology of Rio Grande do Norte (IFRN) for encouraging the improvement of their employees and providing me the opportunity to develop this research.

To my friends of the Petroleum Evaluation and Measurement Laboratory (LAMP/UFRN), for the help with the research and for making the hard times easier, with the coffee breaks and fun moments. To the engineer Berton Meira, for the guidance he provided on the development of this work.

To my advisor, Andres Ortiz Salazar, for the valuable orientation during this research, always sharing his knowledge with patience, humility, and good humor.

To my family, especially my parents, my sister, and my niece – José, Alzira, Juliana, and Manuela – for their continuous support in all my achievements. To my girlfriend and lifemate, Kezia Raphaela, for her complicity, patience, and love.

Abstract

A device known as pipeline inspection gauge (PIG) runs through oil and gas pipelines performing various maintenance operations in the oil and gas industry. The PIG's velocity, which plays a role in the efficiency of these operations, is usually obtained indirectly from odometers installed in it. Although this is a relatively simple technique, the loss of contact between the odometer wheel and the pipeline results in measurement errors. To help reduce these errors, this work employed neural networks to estimate the speed of a prototype PIG, using the pressure difference that acts on the device inside the pipeline and its acceleration instead of using odometers; we built static networks (e.g., multilayer perceptron) and recurrent networks (e.g., long short-term memory). We developed the prototype PIG with an embedded system based on Raspberry Pi 3 to collect speed, acceleration, and pressure data for the model training. The Python library TensorFlow was used to implement supervised neural networks. To train and evaluate the models, we used the PIG testing pipeline available at the Petroleum Evaluation and Measurement Laboratory of the Federal University of Rio Grande do Norte (LAMP/UFRN). Our results show that the models were able to learn the relationship between the differential pressure, acceleration and speed of the PIG. The proposed approach can complement the odometer-based systems, thereby increasing the reliability of speed measurement.

Keywords: Pipeline Inspection Gauge (PIG), Artificial Neural Networks, Embedded Systems, Raspberry Pi.

Resumo

Um dispositivo conhecido como *pipeline inspection gauge* (PIG) percorre oleodutos e gasodutos realizando diversas operações de manutenção na indústria de petróleo e gás. A velocidade do PIG, que desempenha um papel importante na eficiência dessas operações, geralmente é obtida indiretamente a partir dos hodômetros nele instalados. Embora esta seja uma técnica relativamente simples, a perda de contato entre a roda do hodômetro e a tubulação resulta em erros de medição. Para ajudar a reduzir esses erros, este trabalho empregou redes neurais para estimar a velocidade de um PIG protótipo, utilizando a diferença de pressão que atua no dispositivo dentro dos dutos e sua aceleração ao invés de utilizar hodômetros; construímos redes neurais estáticas (por exemplo, *multilayer perceptron*) e redes recorrentes (por exemplo, *long short-term memory*). Desenvolvemos o PIG protótipo com um sistema embarcado baseado em Raspberry Pi 3 para coletar dados de velocidade, aceleração e pressão para o treinamento do modelo. A biblioteca Python TensorFlow foi usada para implementar redes neurais supervisionadas. Para treinar e avaliar os modelos, utilizamos o duto de testes de PIGs disponível no Laboratório de Avaliação e Medição em Petróleo da Universidade Federal do Rio Grande do Norte (LAMP/UFRN). Nossos resultados mostram que os modelos foram capazes de aprender a relação entre a pressão diferencial, a aceleração e a velocidade do PIG. A abordagem proposta pode complementar os sistemas baseados em hodômetros, aumentando assim a confiabilidade da medição de velocidade.

Palavras-chave: Pipeline Inspection Gauge (PIG), Redes Neurais Artificiais, Sistemas Embarcados, Raspberry Pi.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A device known as pipeline inspection gauge (PIG) runs through oil and gas pipelines performing valuable maintenance operations for the oil and gas industry. The PIG's velocity, which plays a role in the efficiency of these operations, is usually obtained indirectly from one or more odometers installed in the PIG. Although this is a relatively simple technique, the slip between the odometer wheel and the pipeline results in measurement errors, so the effort to overcome or reduce the odometer measurement errors is crucial.

1.1 Motivation

PIGs are devices that move inside the ducts, performing from simple cleaning to detailed inspection of pipeline integrity. They can be used at virtually any stage in the life of a pipeline, such as commissioning, maintenance, and decommissioning (Tolmasquim 2004, Russell 2005). These devices are driven by the medium itself (the product transported), due to the differential pressure that acts on the device inside the pipe.

The speed of a PIG is usually calculated from the measurement of the distance employing an odometer attached to the body of the PIG (Lima et al. 2017). The operating principle of the odometer consists of counting the revolutions of a wheel rolling along the duct's surface. First, the distance is measured as a function of the wheel perimeter and the number of revolutions. Then, the speed can be calculated as the derivative of distance

with respect to time.

To control the speed of a PIG, a widely applied technique uses an actuator called a bypass valve, which allows for regulating the differential pressure and, therefore, the speed (Nguyen et al. 2001a). In this case, a closed-loop speed control system generates the signal that controls the bypass valve.

As the PIG's speed is a function of the differential pressure acting on it, which causes the driving force of the PIG, the differential pressure may be used to predict the PIG's speed. So it is necessary to find the relationship between the differential pressure and the PIG speed, that is, to build a mathematical model of the system, which can be obtained through different approaches. Artificial neural networks are often used to build models that are difficult to obtain from physical modeling. Therefore we propose to employ neural networks to build speed prediction models, using the differential pressure that acts on a PIG and its acceleration.

At the Petroleum Evaluation and Measurement Laboratory of the Federal University of Rio Grande do Norte (LAMP/UFRN), several works related to PIGs have been developed (Freitas et al. 2014, Lima et al. 2015, Freitas et al. 2016, Freitas 2016, Lima et al. 2017, Araujo 2017, Lima 2019). Following these works, this thesis aimed to contribute to the advancement of research related to the speed control of PIGs, especially in the matter of speed measurement.

1.2 Contributions

The contributions of this work are summarized as follows:

- Build a prototype PIG with an embedded system that allows measuring and processing variables related to the PIG's dynamic as it travels along a testing pipeline (speed, acceleration, and pressure);
- Propose an approach to predict the PIG's speed from differential pressure and ac-

celeration using artificial neural networks;

- Build and evaluate neural networks to validate the proposed approach using experimental data from the testing pipeline.

1.3 Justification

In both cleaning and inspection tasks, among others, the speed of a PIG should generally be kept constant for better efficiency of the operation (Yardi 2004, Haniffa & Hashim 2012).

Especially in gas pipelines, the need to control the speed of a PIG is even more evident due to the great occurrence of velocity excursions (Haniffa & Hashim 2012). The velocity excursion is a high velocity reached by the PIG in certain operating conditions, a consequence of the high differential pressure that arises when the device has its displacement limited by some obstruction in the pipeline.

Since the speed must be kept constant, the occurrence of velocity excursions must be avoided. In the case of smart PIGs, which have sensors for measuring the integrity of the duct, the abrupt variation in velocity can greatly compromise the data obtained by the various sensors that make up the acquisition system (Yardi 2004). Therefore, the speed controller must be able to reduce the effect of velocity excursions.

Although the use of the odometer allows calculating the speed in a relatively simple way, this technique presents some difficulties related mainly to the loss of contact between the odometer and the duct wall. If the velocity measurement during the occurrence of the velocity excursions is compromised, the action of the controller will also be compromised. Therefore, developing alternative techniques for obtaining speed is crucial in the search for a higher performance of the control system, as well as to assure greater reliability.

1.4 Organization of the work

After this introductory chapter, the rest of the work is organized as follows:

- Chapter 2: A bibliographic review on measuring the distance and speed of PIGs and on employing neural networks as function approximators;
- Chapter 3: A theoretical background of the main topics related to this work;
- Chapter 4: A description of the materials and methods applied;
- Chapter 5: The results and discussion;
- Chapter 6: The conclusion and suggestions for further research.

Chapter 2

Literature review

Next, we briefly describe works directly related to the development of this thesis. They are divided into three sections, according to the contribution area: first, we present works related to the speed control of PIGs; next, we present research related to distance and speed measurement in PIGs; then, we present the use of artificial neural networks, some of their applications, and their use for system identification. The references are organized in chronological order within each section.

2.1 Speed control of PIGs

Several works have been developed on the topic of speed control of PIGs. According to Nguyen et al. (2001a), a PIG is more effective when it moves at a constant speed. The authors derived mathematical models to analyse dynamic characteristics in natural gas pipelines, such as the gas flow and the PIG's position and velocity. The results include the simulation of a velocity excursion event.

Another work of Nguyen et al. (2001b) states that PIGs used for batching, cleaning and liquid removal in gas pipelines travels generally travels along the regular flow of product in the range of 1-5 m/s in liquid pipelines and 2-7 m/s in gas pipelines. For inspection operations, though, the optimal speed range is more defined (e.g., 0.5-4 m/s for corrosion tools).

Yardi (2004) addressed the problem of controlling the speed of a PIG to achieve greater efficiency in cleaning operations. He also stated that smart PIGs must move at a constant speed to avoid distortions in the collected data since the sampling time of the acquisition system is constant. The author presented a history of speed control of PIGs, citing related patents, and described a speed control system that uses a bypass flow valve. The bypass flow was regulated by the controller based on the feedback of a flowmeter and controlled by a motorized butterfly valve.

Haniffa & Hashim (2012) emphasized that speed control is crucial in different PIGs' types, since the efficiency of cleaning and inspection operations are greatly dependent on the PIG's speed. They also described various speed control methods, classifying them as passive or active: in passive methods, the PIG's speed is externally controlled by controlling the pipeline related variables, such as the operating pressure or flow rate; in active methods, the speed of the PIG inside the pipeline is controlled by some internal mechanisms embedded on the device.

Liang et al. (2017) presented a method for active speed control of PIGs with a brake unit, which is a self-regulated device that generates a drag force that slows down the PIG. The authors presented a numerical solution for solving the speed governing equations and simulation results.

2.2 Distance and speed measurement of PIGs

The most used technique to measure the traveled distance of a PIG is employing one or more odometers. Once the distance is obtained, the speed can be calculated. Sadovnychiy & Lopez (2005) proposed a distance measurement correction algorithm to reduce the error caused when the odometer wheel reaches welds seams inside the pipeline. First, the algorithm determined when the PIG passes over a weld seam (disturb) by detecting the odometer's arm displacement using a potentiometer installed on the arm's axis. Next,

it replaced the odometer speed indications in the disturbing section with the estimated speed, which was assumed to be constant for short periods. Finally, it performed double integration of the estimated speed to compute the distance traveled in the disturbing sections. The results were evaluated on a simulator.

According to Sadovnychiy et al. (2006), the odometer's accuracy depends on various factors, such as PIG speed, cleanliness and conditions (size of welds or defects) of the interior of the pipe, and the characteristics of the transported fluid. The authors pointed out that the leading causes of distance measurement errors are slipping and loss of contact between the odometer wheel and the duct surface. Loss of contact can be caused by defects in the duct or arise at derivation points in the duct. They developed a simulator of odometer wheel movement in a pipe and analyzed the error caused by the loss of contact.

Santana et al. (2010) used a nonlinear sensor fusion algorithm based on an extended Kalman filter (EKF) to estimate the trajectory of PIGs. The algorithm combined data from a low-cost IMU (acceleration and angular rate), an odometer (speed), and topographic landmarks (distance). Instead of using an actual PIG moving through a pipeline loop, the authors used an automobile along closed trajectories to perform preliminary experiments. Using only the low-cost IMU, it was not possible to reconstruct the traveled path. However, the performance significantly improved when combining the IMU and additional speed measurements from the odometer and position measurements from the topographic landmarks.

Money et al. (2012) described a cleaning PIG with a speed control system. Up to three odometers were used to log the distance covered by the PIG and compute speed to the control system (the fastest odometer is automatically chosen). The device also has differential pressure, acceleration, and angular rate sensors – however, the authors did not describe what these sensors were used for in this device.

According to Zhu et al. (2016), due to its construction characteristics, the odometer is naturally prone to cumulative measurement errors, and the main cause of these errors

is the pipe welds. In the authors' tests in an experimental rig, the error varied with the odometer's speed. The measured distance was greater than the actual distance when the speed was small since the arc length of the weld is longer than the width; the distance was lower when the speed was high due to the loss of contact between the pipe wall and the odometer wheel in this condition.

Precisely locating the defects that a smart PIG has detected along the pipeline is a significant concern. Hence tactical-grade inertial measurement units (IMUs) are used to reconstruct trajectories of the PIG. According to Sahli & El-Sheimy (2016), these IMUs are accurate but also expensive and large devices, which limit their use in pipelines with diameters below 8" or less. An alternative is to use a micro-electromechanical system (MEMS) IMU, which has lower performance but is cheaper and smaller. The authors addressed the issue of aiding a MEMS-based inertial navigation system to replace tactical-grade IMUs. They described a new methodology for using MEMS IMUs employing an extended Kalman filter (EKF) and the pipeline junctions to increase the position measurement accuracy.

Araújo et al. (2018) proposed a model that employs neural networks to obtain the relationship between the differential pressure and the speed of a PIG in a testing pipeline. The training set consisted of speed data (calculated from the PIG odometer) and differential pressure (measured by sensors installed along the pipeline). Upon PIG retrieval, the neural network predicted the speed using data recorded during the run.

Zhu et al. (2019) carried out experiments to determine the odometer trajectory on a test bench. A high-speed camera recorded the odometer's behavior when passing over a weld, allowing a detailed analysis of its trajectory. The results showed that changes in the spring force, size, and material of the odometer can improve its accuracy. They also stated that the slower the PIG speed, the greater the accuracy of the odometer in their experiments.

2.3 Artificial neural networks

In the research developed by Narendra & Parthasarathy (1990), Sjöberg et al. (1994) and Sjöberg (1995), the authors demonstrated that artificial neural networks (ANNs) have been widely researched and used in the context of systems identifications, particularly in the modeling of non-linear systems.

The work of Habtom (1998) pointed out the ability that neural networks have to learn relationships that are difficult to obtain from physical modeling. The author described the application of neural networks in developing models for predicting process variables, emphasizing recurrent networks in modeling systems that involve some temporal relationship.

According to Haykin (2001), artificial neural networks (ANNs) have important properties, such as generalization, robustness, adaptability, intrinsic non-linearity, and input-output mapping. These properties make neural networks candidates for solving several problems, such as image processing, control and identification of dynamic systems, and pattern classification.

The work of Ferrari & Piuri (2003) extensively addressed the applications of artificial neural networks in the area of measurement systems. According to the authors, neural networks can be used in soft sensors, modeling, fusion, fault diagnosis, and calibration applications.

According to Fortuna et al. (2006), the use of neural networks is one of the main approaches used to build soft sensors, which are dynamic models devoted to the estimation of plant variables. The author pointed out that neural networks are becoming standard tools for developing non-linear soft sensors due to the good performance obtained for many real-world applications and the availability of software tools that help the designer.

In their review work, Abiodun et al. (2018) described various neural network architectures and cited diverse areas of application, including speech recognition, computer

vision, identification and control, medical diagnosis, signal processing, and weather forecast.

2.4 Chapter summary

Due to the importance of the PIGs' speed for the efficiency of their operations, many author have authors have been developing control systems aiming to keep a near constant speed. The speed can be controlled externally by regulating the operation pressure or flow rate of the pipeline (passive method) or by an embedded system installed on the PIG (active method). To measure the speed, the most usual technique is based on the use of odometers. However, odometers present measurement errors due to slip and loss of contact between the wheel and the duct surface. The error varies depending on the mechanical parameters of the odometer-based measurement system (spring force and wheel material), the duct condition, and the PIG's velocity. Therefore, different techniques were proposed to minimize the problem, contributing to more accurate distance and speed measurement.

Chapter 3

Theoretical background

We present some basic concepts and theoretical aspects related to the development of the work: PIG motion dynamics, inertial sensors, machine learning basics, and artificial neural networks.

3.1 Pipeline inspection gauges (PIGs)

Pipeline inspection gauges (PIGs) are devices that move inside the ducts and are capable of performing from simple cleaning to detailed inspection of pipeline integrity. Figures 3.1, 3.2, and 3.3 show examples of commercial PIGs designed for different purposes.

Figure 3.1: Example of cleaning PIG. Source: Modified from *Batching Pigs* (2019).

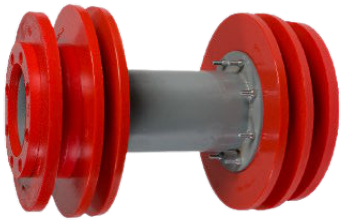


Figure 3.2: Example of sealing PIG. Source: Modified from *Batching Pigs* (2019).

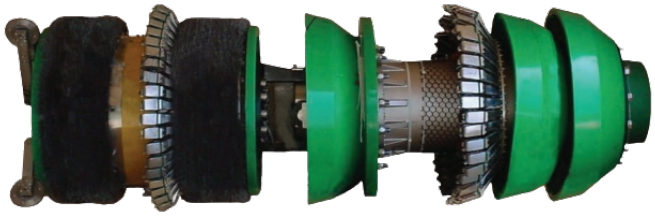


Figure 3.3: Example of smart PIG. Source: Modified from *Pipeway International* (2008).

PIGs can be classified basically into three groups: cleaning, sealing, and smart PIGs. Their names are given according to the function to be performed:

- Cleaning PIGs – as the name suggests, they can clean unwanted residues inside the pipeline;
- Sealing PIGs (batching PIGs) – they isolate different sections of the duct in order to promote, for example, the separation of different products in the same pipeline;
- Smart PIGs – also known as instrumented PIGs or in-line inspection (ILI) tools, they have an embedded system composed of sensors capable of locating defects such as dents, crack, and corrosion in the pipeline.

3.2 PIG motion dynamics

A PIG travels in the pipeline through the transported fluid, thanks to the differential pressure that acts on the PIG, as illustrated in Figure 3.4.

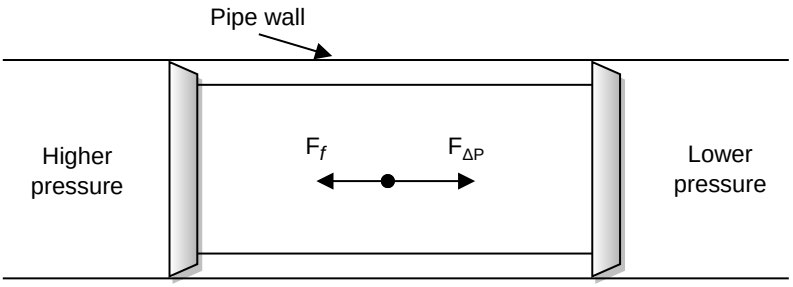


Figure 3.4: Forces involved in the PIG motion.

Considering the duct parallel to the ground, which implies zero net force on the PIG in the vertical direction, a simplified dynamical model can be obtained from Newton's Second Law, based on Nieckele et al. (2001), as follows:

$$m\frac{dv_{pig}}{dt} = F_{\Delta P} - F_f \tag{3.1}$$

$$F_{\Delta P} = A\Delta P \tag{3.2}$$

$$F_f = F_c v_{pig} \tag{3.3}$$

$$m\frac{dv_{pig}}{dt} = A\Delta P - F_c v_{pig} \tag{3.4}$$

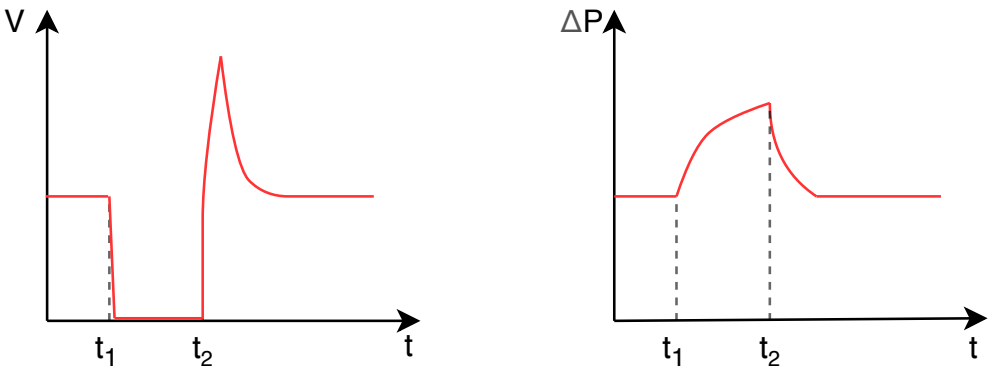
Where: m is the mass of the PIG; v_{pig} is the velocity of the PIG; A is the cross-sectional area of the PIG's rear; $F_{\Delta P}$ is the driving force; F_f represents the friction force; ΔP is the differential pressure that acts on the PIG; F_c is the axial contact force between the PIG and the pipe wall.

The dynamic behavior of a PIG inside the pipe is described by its dynamic equation, coupled with the fluid's governing equations. However, the analysis of equation 3.4 is sufficient to state that the acceleration of the PIG, and therefore its velocity variation, is directly proportional to the differential pressure.

Eventually, some obstacle impedes the PIG's motion inside the pipeline, like debris from the transported product. This situation can lead to the occurrence of the so-called velocity excursion, which is explained below.

Velocity excursions

When an obstacle inside the duct prevents the movement of the PIG, the upstream pressure increases significantly in relation to the downstream pressure. The differential pressure (ΔP) reaches the point where the device can overcome the obstruction, causing a phenomenon known as velocity excursion, which is a high velocity (V) reached by the PIG in these conditions. This behavior is illustrated in figures 3.5a and 3.5b: the PIG stops at the instant t_1 , and the differential pressure starts increasing from that very moment; at the instant t_2, as the differential pressure becomes high enough to make the PIG overcome the obstacle, the velocity excursion occurs.



(a) PIG's velocity.

(b) Differential pressure acting on the PIG.

Figure 3.5: Behavior of velocity (V) and differential pressure (ΔP) with respect to time (t) in the presence of a velocity excursion.

3.3 Machine learning basics

3.3.1 Artificial intelligence and machine learning

Artificial intelligence is a computer science field, born in the 1950s, dedicated to the effort of automating intellectual tasks usually performed by humans. It encompasses machine learning and deep learning, but also other approaches (Chollet 2017).

Machine learning is a subfield of artificial intelligence that can be defined as the "field of study that gives computers the ability to learn without being explicitly programmed." This definition is credited to the American computer scientist Arthur Samuel. Mitchell (1997) provides a more formal definition:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

Considering the context of our work, we can use this definition to offer the following example:

- Task T: estimating PIG's velocity using differential pressure;
- Performance measure P: the difference between the estimated velocity and true velocity;
- Training experience E: differential pressure and velocity measurements.

3.3.2 Learning processes

According to Haykin (2001), learning processes are categorized into: supervised learning, unsupervised learning, semisupervised learning, and reinforcement learning. In supervised learning, a set of inputs (features) is presented to the machine along with a set of the respective desired outputs (labels or targets), so it can be said that there is a figure of a teacher who teaches the machine how it should behave.

In contrast, in unsupervised learning, the machine is trained to learn only from input data. In this case, there is no teacher, i.e., no labeled data for training the machine.

In semi-supervised learning, there are typically a small amount of labeled data with a large amount of unlabeled data, so this category combines the supervised and unsupervised approaches.

Finally, in reinforcement learning, the machine's response is iteratively evaluated to positively or negatively reinforce its behavior according to some performance index. In this work, we used the supervised approach.

3.3.3 Training, test, and validation sets

The data used to build a supervised machine learning model is usually split into training and test sets. As the name suggests, the training set is used to train the model. Once the model is trained, i.e., the learning process has finished, the test set is used to test the model's performance. In addition, a validation set is often drawn from the training set to help adjust the model's hyperparameters.

3.3.4 Hyperparameter tuning

Hyperparameters are values that control a model's structure or some settings of the learning algorithm but are not tuned (adjusted) by the learning algorithm itself. Some of these hyperparameters affect the time and memory cost of training and running the model, while others affect its generalization capability.

In the case of the neural networks used to build the models developed in this work, examples of hyperparameters are the number of layers, the number of neurons on each layer, and the activation functions of each layer.

Two basic approaches to adjusting the hyperparameters are manual tuning and automatic tuning. Manual tuning essentially requires understanding the influence of the

hyperparameters on the model's performance. Automatic tuning is often much more computationally expensive, but the need to understand the hyperparameters' influence is significantly reduced (Goodfellow et al. 2016).

Grid search and random search are the most widely used strategies for automatic tuning (Bergstra & Bengio 2012). In grid search, for each hyperparameter the user defines a finite set of values to explore (e.g., number of neurons and activation functions in a neural network's hidden layer), then the search algorithm trains a model for every combination of hyperparameters on the search space defined by the user. In random search, the search algorithm chooses a given number of combinations of random values (within ranges specified by the user) for each hyperparameter.

3.3.5 Overfitting, underfitting, and capacity

According to Goodfellow et al. (2016), the major challenge in machine learning is that the model must perform well beyond the data it was trained (training set), i.e., on previously unseen data. This desired ability to perform well on previously unseen data (test set) is called generalization.

When the model cannot perform well on the training set, the model is said to suffer from underfitting. Conversely, overfitting occurs when the model performs well on the training data but performs poorly on the test data, presenting a large gap between the training error and the test error.

A model is more likely to overfit or underfit by altering its capacity, which can be informally defined as the model's ability to approximate functions. Models with low capacity may be unable to fit the training set. Models with high capacity can approximate more complex functions but are prone to overfit by memorizing useless properties of the training data set that might compromise the generalization capability (Goodfellow et al. 2016).

3.4 Artificial neural networks

Haykin (2001) describes an artificial neural network [1] (ANN) as a distributed system, inspired by the human brain, composed of simple processing units called neurons, which have a natural propensity for acquiring and storing knowledge. Figure 3.6 shows a neuron model.

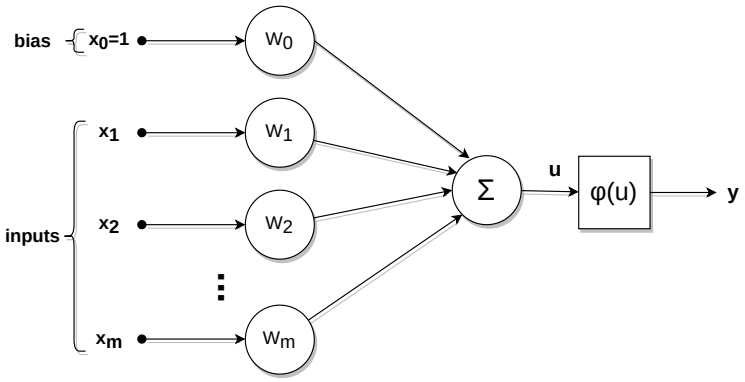


Figure 3.6: Artificial neuron model.

This model can be described mathematically by equations 3.5 and 3.6:

$$u = \sum_{i=0}^{m} w_i x_i \tag{3.5}$$

$$y = \varphi(u) \tag{3.6}$$

Where: The fixed input x_0 with weight w_0 is called bias; $x_1, x_2, ..., x_m$ are the $m-1$ inputs; $w_1, w_2, ..., w_m$ are parameters called synaptic weights; u is the linear combination of the inputs plus the bias; φ is the activation function; y is the neuron output.

The activation function defines the output of a neuron in terms of u. The following activation functions are typically used:

- Sigmoid

$$\varphi(u) = \frac{1}{1+e^{-u}} \tag{3.7}$$

[1] The terms *artificial neural network*, *neural network*, *neural net*, or simply *network* are used interchangeably throughout this work.

- Hyperbolic tangent

$$\varphi(u) = \frac{e^{u} - e^{-u}}{e^{u} + e^{-u}} \tag{3.8}$$

- Linear

$$\varphi(u) = u \tag{3.9}$$

- Rectified Linear Unit (ReLU)

$$\varphi(u) = \begin{cases} 0, & \text{if } u < 0 \\ u, & \text{if } u \geq 0 \end{cases} \tag{3.10}$$

3.4.1 Learning process of neural networks

The learning process of a neural network consists of adjusting the synaptic weights and bias of the neurons to minimize a cost function. The algorithm that performs this function is often called optimizer, and the gradient-descent and its variants are between the most used optimizers. The ability to learn the behavior of a system from a limited set of samples is one of the main characteristics of neural networks. Once the network has been trained, it is ideally able to produce an adequate output from any signals applied to its inputs, that is, it is able to generalize solutions (Silva et al. 2010).

3.4.2 Architectures of neural networks

The architecture of an artificial neural network defines how its neurons are connected. Fundamentally, three distinct neural network architectures can be identified: single-layer feedforward networks, multilayer feedforward networks, and recurrent neural networks (Haykin 2001). In this thesis, we employed multilayer feedforward networks and recurrent networks.

Feedforward networks

In feedforward networks, neurons are organized in layers, and information flows unidirectionally from the input to the network output (hence the term feedforward). In single-layer feedforward networks, there is only one layer of neurons, which constitutes the network's output. We do not count the input layer since no computation is performed there. Figure 3.7 represents this type of architecture. Neurons are represented by circles; arrows represent the connections between neurons; x_1, x_2, and x_3 are the network inputs; y_1, y_2, and y_3 are the outputs;

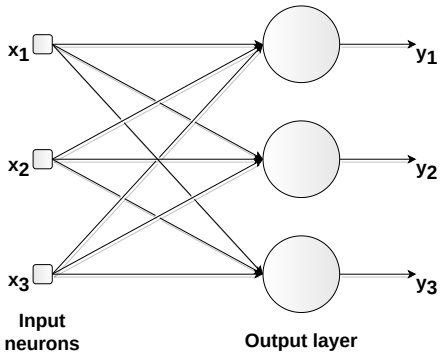


Figure 3.7: Example of single-layer feedforward network with three inputs (x_1, x_2, and x_3) and three outputs (y_1, y_2, and y_3).

In multilayer feedforward networks, the network comprises one or more hidden layers, whose corresponding neurons are called hidden neurons (Figure 3.8). They are given this name because the neurons in the hidden layers are not directly visible from either the network's input or output. A multilayer perceptron (MLP) is a typical multilayer feedforward network.

Recurrent neural networks

Networks with recurrent architecture, also known as feedback networks, are made up of neurons whose outputs are used as inputs to the network itself (Figure 3.9). The introduction of feedback enables recurrent networks to dynamically process information,

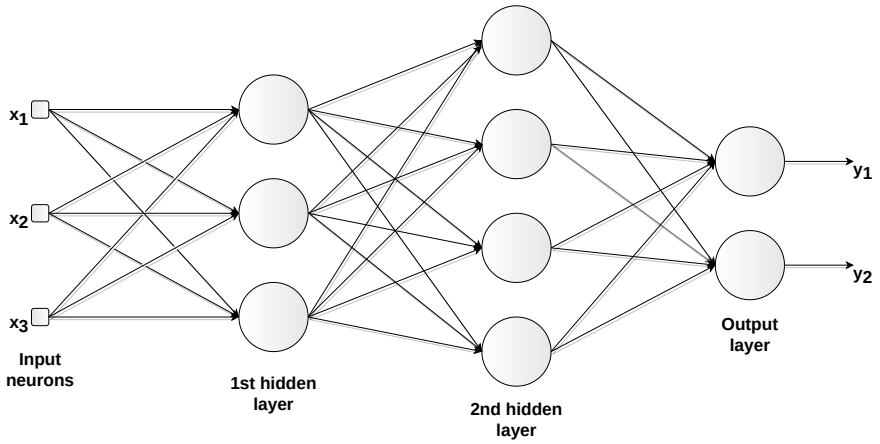


Figure 3.8: Example of multilayer feedforward network with three inputs (x_1, x_2, and x_3), two hidden layers, and two outputs (y_1 and y_2).

allowing their use in applications such as time series forecasting, process control, and systems identification (Silva et al. 2010). Examples of recurrent neural networks are the nonlinear autoregressive network with exogenous inputs (NARX) and the long short-term memory (LSTM).

3.5 Artificial neural networks and system identification

The use of mathematical models is inherent in the most diverse fields of engineering, as they are fundamental to better understand the behavior of a system, in addition to enabling computer simulations. Mathematical modeling can be defined as the area of knowledge that studies techniques for obtaining mathematical models of real systems. A mathematical model is an analog that aims to represent some of the characteristics observed in the real system, such as, for example, its dynamic behavior (Aguirre 2015).

There are several techniques for building a mathematical model and even different models for the same system. These techniques are usually grouped according to the following approaches:

- White-box – also known as physical modeling or first-principles modeling, it consists of building the model from the analysis of the physical phenomena involved in

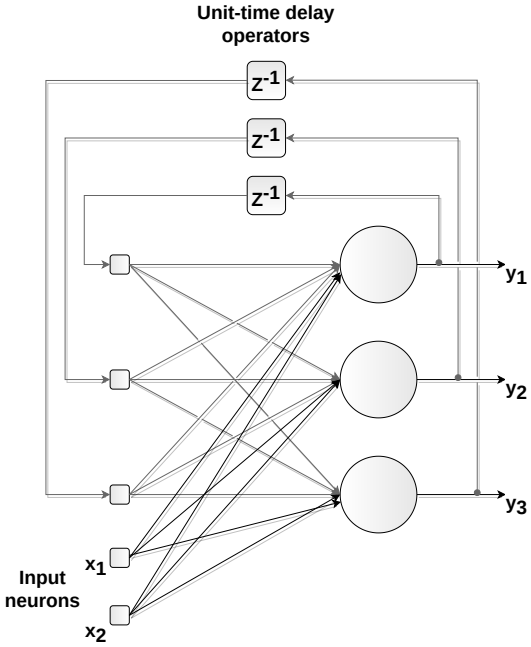


Figure 3.9: Example of recurrent network with two inputs (x_1 and x_2) and three outputs (y_1, y_2, and y_3). The time-delayed outputs are used as inputs to feedback the network.

the system to be modeled;

- Black-box – also called system identification, consists of obtaining the model only from experimental data of the system.
- Gray-box – it is situated between physical modeling and systems identification and, therefore, combines both information related to the physical phenomena and the experimental data of the system.

One of the most significant benefits of the black-box approach is that only a minimal knowledge of the process is required. In contrast, a good understanding of the physical phenomena involved in the process is critical to the development of white-box models (Tangirala 2015). An artificial neural network is essentially a black-box modeling tool, often used to perform non-linear mapping of the input and output of a system.

In the case of the so-called dynamic systems, assuming any instant of time, the output depends not only on the present input but also on its past values (Aguirre 2007). In the following sections, we present three candidate networks concerning temporal processing:

time-delay neural network (TDNN), non-linear autoregressive network with exogenous inputs (NARX), and long short-term memory (LSTM).

3.5.1 Time-delay neural network (TDNN)

Time may be incorporated into a feedforward neural network using time-delayed inputs, in a structure known as time-delay neural network (TDNN), illustrated in Figure 3.10. A TDNN implements a function given by

$$y(n) = F[x(n), x(n-1), \ldots, x(n-p)], \tag{3.11}$$

where: F is a non-linear function; $y(n)$ is the system's response; $x(n)$ is the present value of the input signal; $x(n-1), \ldots, x(n-p)$ are the p past values of the input signal.

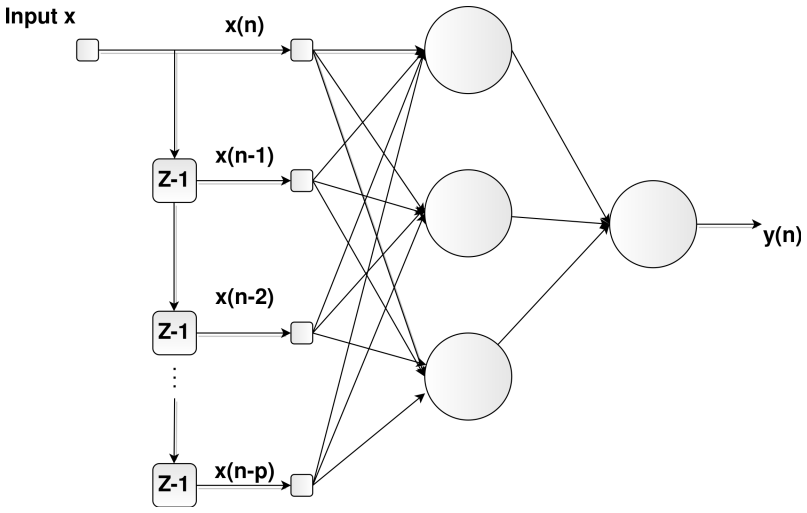


Figure 3.10: Time-delay neural network (TDNN).

3.5.2 Non-linear autoregressive network with exogenous inputs (NARX)

Another approach for temporal processing is to incorporate recurrence in a feedforward neural network by using time-delayed inputs and time-delayed outputs. This struc-

ture is known as non-linear autoregressive network with exogenous inputs (NARX). The NARX model can be expressed by

$$y(n) = F[x(n), x(n-1), \ldots, x(n-p), y(n-1), \ldots, y(n-q)], \tag{3.12}$$

where: F is a non-linear function; $y(n)$ is the system's response; $x(n)$ is the present value of the input signal; $x(n-1), \ldots, x(n-p)$ are the p past values of the input signal; $y(n-1), \ldots, y(n-q)$ are q past outputs.

The NARX model can be trained using the parallel (closed-loop) or the series-parallel (open-loop) configuration. In the parallel configuration, the estimated value of the output (target) is fed back into the model (Figure 3.12). In contrast, in the series-parallel configuration, the true value of the output is used instead of feedbacking the estimated output (Figure 3.11).

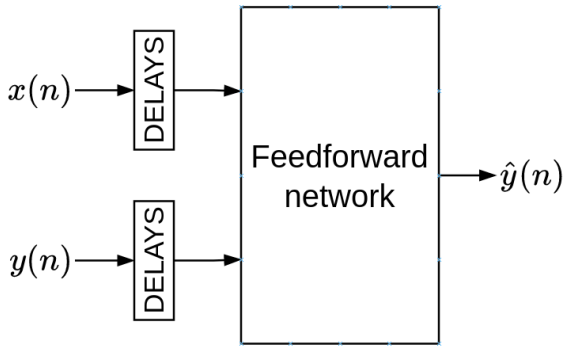


Figure 3.11: Series-parallel (open-loop) configuration.

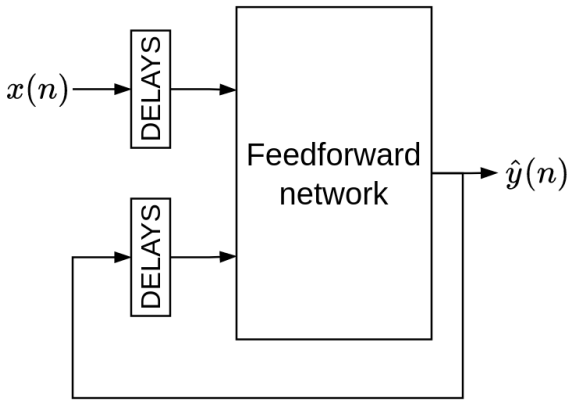


Figure 3.12: Parallel (closed-loop) configuration.

3.5.3 Long short-term memory (LSTM)

The long short-term memory (LSTM) is a recurrent neural network first developed by (Hochreiter & Schmidhuber 1997). It can be represented as shown in Figure 3.13.

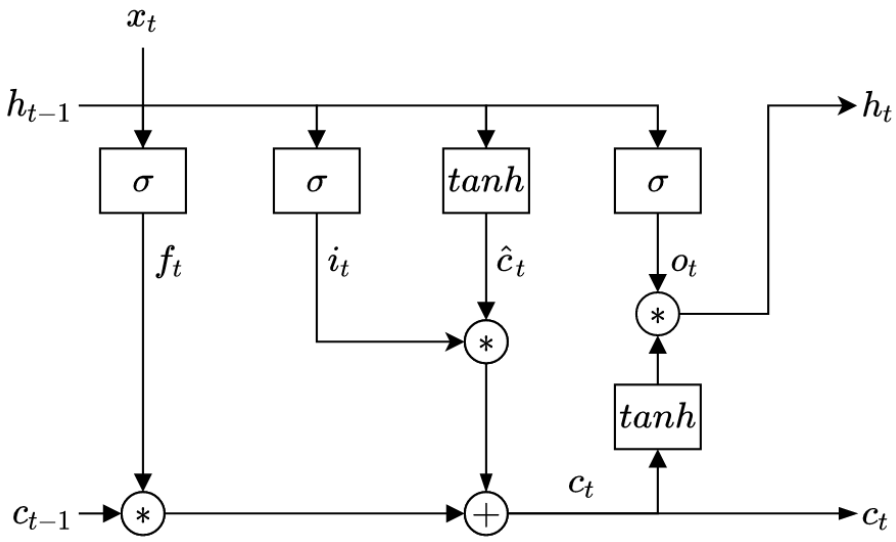


Figure 3.13: Representation of the LSTM cell. x_t is the input vector, c_{t-1} is the previous cell state, h_{t-1} is the previous hidden state, c_t is the current cell state, and h_t is the current hidden state (output).

The LSTM aims to solve the vanishing gradient, gradient explosion, and insufficient long-term ability problems of traditional recurrent neural networks using controllable gates (Liu et al. 2019). These gates are: the forget gate f_t, the input gate i_t, and the

output gate o_t. The LSTM model is described by the following equations:

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \tag{3.13}$$

where: σ is the sigmoid function; x_t is the input vector; h_{t-1} is the previous hidden state; W_{xf} and W_{hf} are the weight vectors of x_t and h_{t-1} on input gate, respectively; b_f is the bias of the input gate.

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \tag{3.14}$$

where: W_{xi} and W_{hi} are the weight vectors of x_t and h_{t-1} on input gate, respectively; b_i is the bias of the input gate.

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \tag{3.15}$$

where: W_{xo} and W_{ho} are the weight vectors of x_t and h_{t-1} on output gate, respectively; b_o is the bias of the output gate.

$$\hat{c_t} = tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{3.16}$$

where: $\hat{c_t}$ is an intermediate state; $tanh$ is the hyperbolic tangent function; W_{xc} and W_{hc} are the weight vectors of x_t and h_{t-1} on the intermediate state, respectively; b_c is the bias of the intermediate state.

Finally, the current cell state c_t and the current hidden state h_t (output) are given by:

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \hat{c_t} \tag{3.17}$$

$$h_t = o_t \cdot tanh(c_t) \tag{3.18}$$

3.6 Chapter summary

The displacement of a PIG along the pipeline is due to the differential pressure that acts on it. A simplified mathematical model for its displacement was presented, which relates the differential pressure to the device's speed. In addition, a phenomenon known as velocity excursion was described, originating from the increase in pressure upstream of the PIG when some obstruction in the pipeline prevents its motion. Some basic concepts related to the modeling of dynamical systems, machine learning, and artificial neural networks were also presented.

Chapter 4

Materials and methods

This chapter introduces the materials and methods used to implement the proposed work. We present the prototype pipeline inspection gauges (PIGs), the embedded system and sensors, the PIGs' testing pipeline, the software tools, and the data collection and preparation aspects.

4.1 The prototype PIGs

Initially, we considered to employ the PIG developed in the work of Lima (2019) for the current work. It has two polyurethane supports (cups) with a diameter of 6", a carbon steel capsule with a diameter of 4.6" on the central part of the structure (body), and an odometer installed on the rear (Figure 4.1).

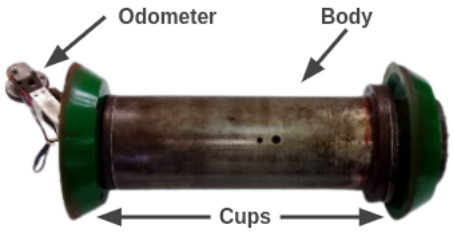


Figure 4.1: Prototype PIG 1. Source: modified from Lima (2019).

This PIG includes an Arduino-based embedded system and a bypass valve for velocity control, actuated by an electro-pneumatic valve. The author's thesis describes the

development, the components, and the assembly of the device.

For the current work, we performed some modifications on the first prototype, such as (a) development of a new embedded system, based on a Raspberry Pi board, aiming to increase the computational power and communication capabilities of the system; (b) removal of the electro-pneumatic valve to improve space utilization; (c) closing of the bypass orifices and sealing improvements (Figures 4.2 and 4.3), since the gauge pressure sensors require that the PIG body is watertight.

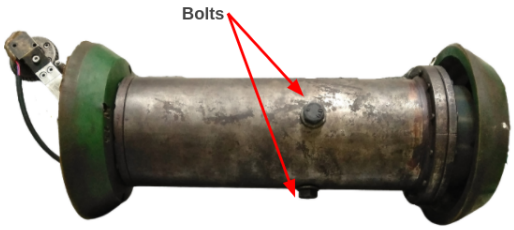


Figure 4.2: Prototype PIG 1. Bolts were installed on the orifices that were used initially for flow bypass.

Figure 4.3: Prototype PIG 1. Gasket sheets were installed on the flanges of the PIG to improve sealing.

However, although some sealing improvements have been achieved, they could not prevent a significant air flow into the PIG body. So after many unsuccessful attempts to solve the leakage problems, we decided to develop a new PIG (Prototype PIG 2) considering mainly the following requirements: improved sealing capability, to prevent air leakage into the PIG body; adequate internal dimensions to install the embedded system

and the sensors; low-cost materials. The new PIG body has two polyurethane supports (cups) with a diameter of 6" and a carbon steel body with a diameter of 4.6" (as in the previous prototype). Figures 4.4, 4.5, 4.6a, 4.6b show an exploded view of the new PIG, the side view, the front view and the rear view of the Prototype PIG 2, respectively.

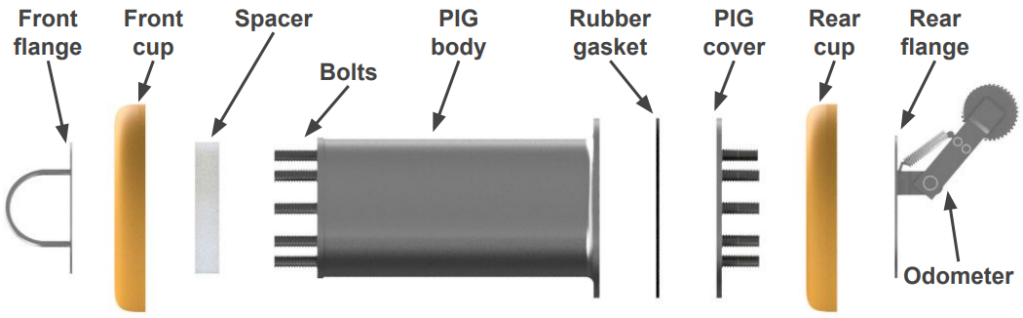
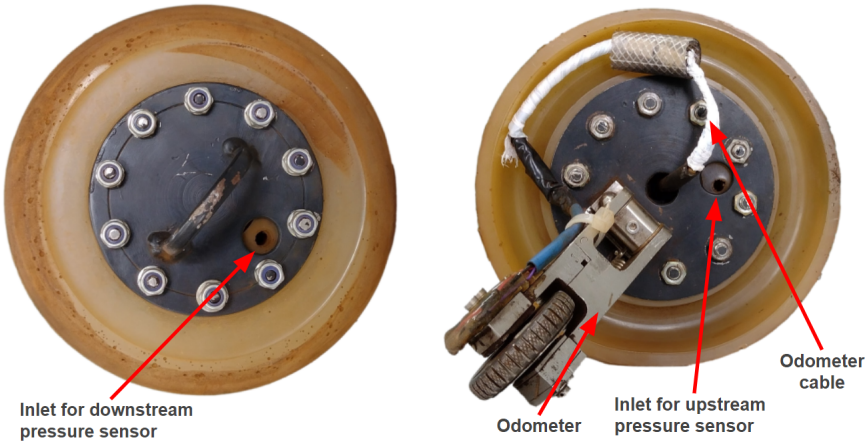


Figure 4.4: Exploded view of the Prototype PIG 2.

Figure 4.5: Prototype PIG 2 (side view picture).



(a) Front view. (b) Rear view.

Figure 4.6: Front and rear views of Prototype PIG 2.

PIG air leakages

The main issue associated with the sealing problems of the PIG is the inadequate functioning of the pressure transducers. The transducers embedded in the PIG measure gauge pressure, that is, in relation to atmospheric pressure (P_{ATM}). Hence, when the pipeline is depressurized (pressure inside the duct equal to P_{ATM}), the pressure indicated by the sensor will be negative if the inside of the PIG is pressurized, as illustrated in Figure 4.7.

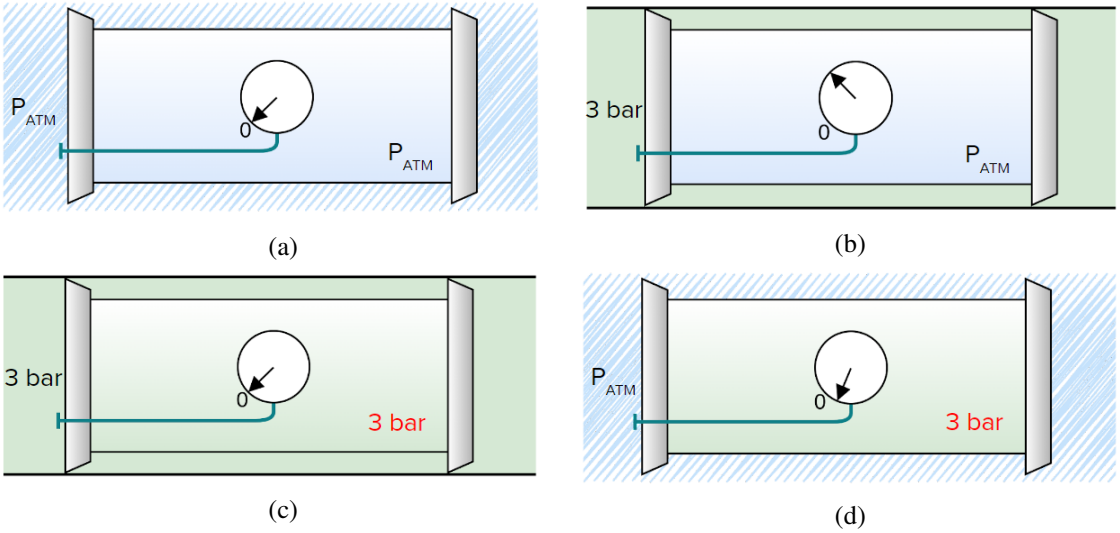


Figure 4.7: Pressure sensor behavior in the absence (a and b) or presence (c and d) of leakages. In (a), before the PIG is inserted into the pipeline, the PIG's internal pressure is equal to the atmospheric pressure (P_{ATM}); in (b), the PIG is inside the pressurized pipeline; in (c), the PIG's internal pressure is equal to the pipeline's pressure; in (d), the PIG is removed from the pipeline, but internal pressure is still pressurized.

The pipeline was pressurized and depressurized while the PIG collected the pressure data inside of it to verify the proper behavior of pressure sensors. Figure 4.8 (page 32) shows these data. First, the pipeline was pressurized up to 4.5 bar (i); After the pressurization, the pressure decreased due to pipeline leakages (ii); The pipeline was then depressurized (iii); After depressurization, the PIG was removed from the pipeline; Finally, the lid of the PIG was opened (v). The negative pressure (highlighted in the figure) indicates that the PIG was pressurized. When opening the lid of the PIG after the run, we actually verified that the PIG had compressed air inside.

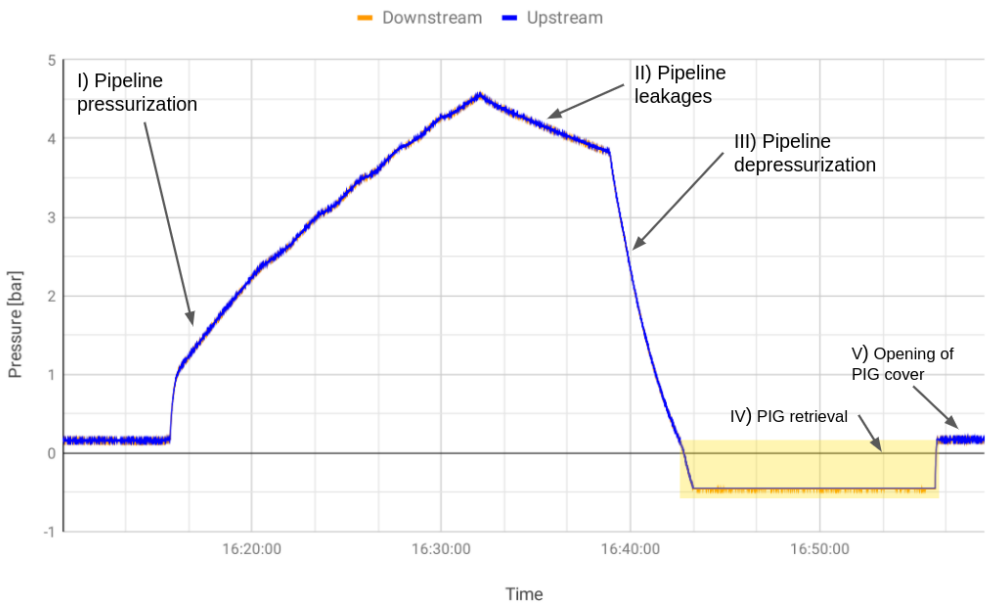


Figure 4.8: Inside the pipeline, the PIG measured pressures during an experiment to verify the functioning of the pressure sensors. The negative pressure (highlighted) occurs due to the PIG's sealing issues.

Figure 4.9 shows data from the pressure sensors collected in a further experiment. In this case, no negative pressures were observed. This is the desired behavior of the sensors.

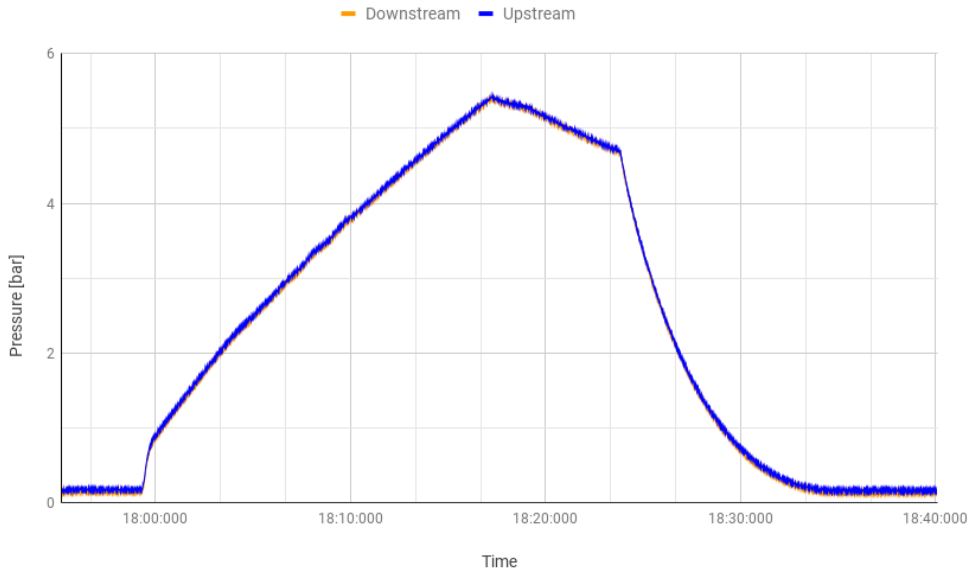


Figure 4.9: Inside the pipeline, the PIG measured pressures during an experiment to verify the functioning of the pressure sensors. In this case, the sensors worked correctly.

4.2 Embedded system and sensors

The core element of the embedded system is a Raspberry Pi 3 Model B+, as it is used for the acquisition, storage, and processing of the sensors' data. The sensors measure the following variables: distance, pressure, and acceleration. Figure 4.10 shows an overall representation of the system.

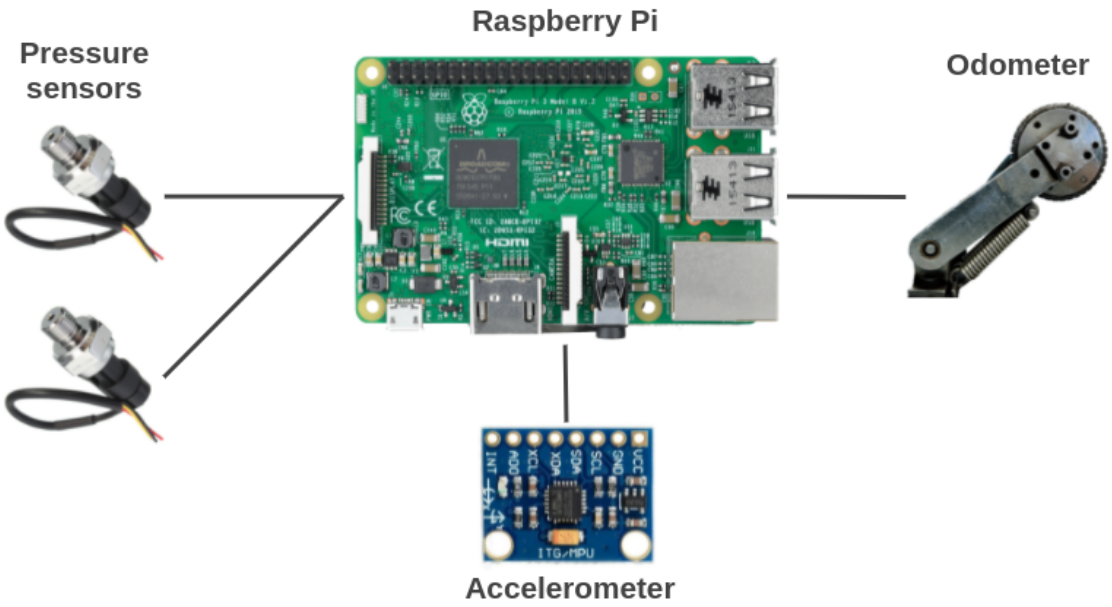


Figure 4.10: An overall representation of the embedded system's elements.

Attached on top of the Raspberry Pi is an auxiliary electronic board that we named Pi Add-On Board, which is an auxiliary board constructed from a universal prototype printed-circuit board that we developed to interface the Raspberry Pi with the pressure sensors and the odometer. Figures 4.11 and 4.12 show the top and bottom views of the Add-On Board, respectively. The complete electronic schematic of the Pi Add-On Board and other hardware elements of the embedded system is presented in Atppendix B.

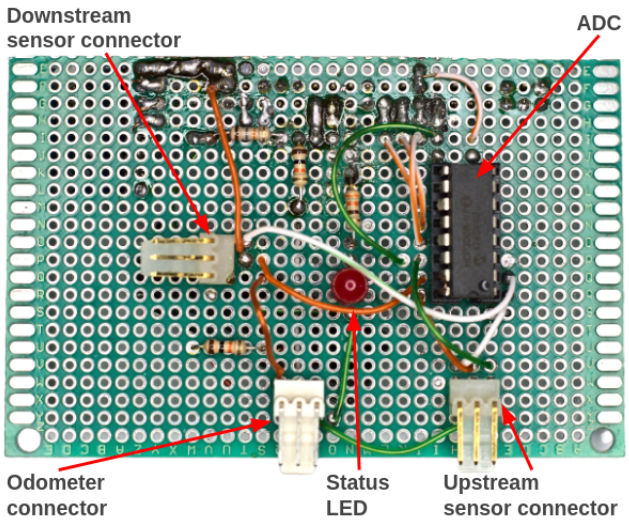


Figure 4.11: Top view of the Pi Add-On Board. An analog-to-digital converter (ADC) was used to interface the pressure sensors with the Raspberry Pi.

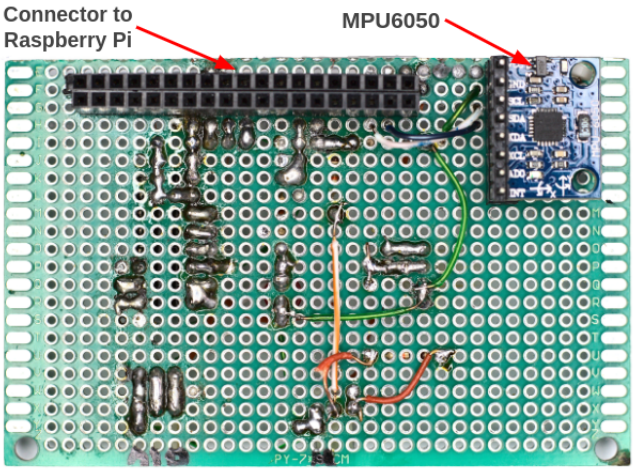


Figure 4.12: Bottom view of the Pi Add-On Board. The accelerometer (MPU6050) was mounted on the Pi Add-On Board.

While the Raspberry does not have a built-in analog-to-digital converter (ADC), the pressure sensors' output is an analog voltage signal, so an external ADC was required to read the pressure from the sensors. The ADC MCP3008 was used, featuring a resolution of 10 bits, 8 input channels, and compatibility with the Serial Peripheral Interface (SPI) protocol. In the odometer case, the output is a digital voltage signal that is either 0 V (low) or 5 V (high). Since the maximum voltage for the Raspberry's input is 3.3 V, the output

of the odometer had to be correctly conditioned, i.e., reduced from 5 V to approximately 3.3 V. This was accomplished in the current work with the voltage divider presented in Figure 4.13. The resistance of R1 and R2 are, respectively, 10 kΩ and 18 kΩ; given a supply voltage of 5 V, it allowed to reduce the output voltage to 3.2 V.

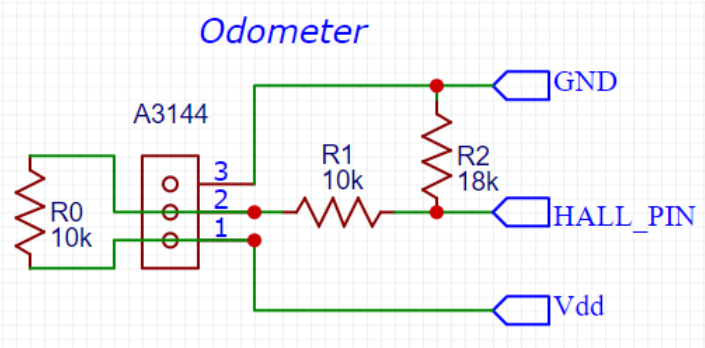


Figure 4.13: Voltage divider used to reduce the voltage of the odometer's output signal.

Still in Figure 4.13, pin 2 is the output of the odometer; HALL_PIN is the input port of the Raspberry Pi that reads the odometer signal; GND is the ground pin; VDD is the voltage source. The resistor R0 (10 kΩ) between pins 1 and 3 of the odometer connector is required by the A3144 Hall-effect switch.

4.2.1 Raspberry Pi 3 Model B+

The Raspberry Pi is a single-board computer (SBC), a digital computer with all the components necessary for its operation – such as microprocessor, input and output (I/O) interfaces, memory, and network interfaces – located on a single printed circuit board. Since the launch of the Raspberry Pi in 2012, low-cost SBCs have become quite popular. They have been used for diverse purposes, such as low-cost personal computers, file servers, media centers, Internet of Things (IoT), robotics, and home automation (Barnatt 2019). Table 4.1 summarizes some features of the SBC used in this work, the Raspberry Pi 3 B+.

Operating system	Raspbian GNU/Linux 10 (buster)
Processor	Cortex-A53 (ARMv8) 64 bits quad-core
Clock	1,4 GHz
RAM memory	1 GB
I/O interface	40 GPIO pins
Communication	Bluetooth 4.2, IEEE 802.11 5 GHz, Gigabit Ethernet
Dimensions	85 x 56 x 17 mm

Table 4.1: Main features of the Raspberry Pi used in this work (Raspberry Pi 3 Model B+). Source: Pajankar (2017).

4.2.2 Pressure sensors

Two pressure sensors were installed on the PIG, one to measure the pressure upstream (behind) and the other to measure the pressure downstream (ahead) of the device. The working principle of the sensors is the Hall effect. Figure 4.14 shows a simplified diagram of the internal construction of the instrument. A bellow with a magnet is placed to move closer to a Hall-effect sensor when the pressure increases since the closer the bellow is to the sensor, the higher is the magnetic field. Figure 4.15 shows the sensor used and table 4.2 presents some features of the instrument.

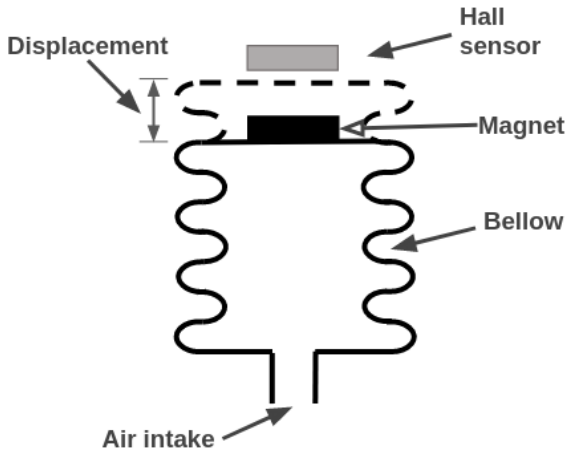


Figure 4.14: Working principle of the pressure sensor. The hall sensor is fixed, while the magnet moves according to the applied pressure.

Figure 4.15: Pressure sensor.

Feature	Description
Working principle	Hall effect
Pressure range	0 - 5 bar
Output voltage	0.5 - 4.5 VDC
Supply voltage	5 VDC
Response time	2.0 ms
Measurement Accuracy	$\pm$ 1.5 % FS (75 mbar)

Table 4.2: Features of the pressure sensors. Source: Studio (2021).

Curve fitting

A curve fitting process was performed to verify the relationship between the pressure and the output voltage of the sensors. This process involves making a certain number of pressure (independent variable) and voltage (dependent variable) observations, then finding a curve that describes the relationship. The voltage of each transducer was measured for six different pressure values (Table 4.3).

Figure 4.16 shows the devices used to perform the pressure and voltage measurements. The pressure regulator and the manometer were used to control and measure the pressure applied to the sensors, while the multimeter was used to measure the output voltages of the transducers. These voltages were compared with the values provided by the embedded system.

Pressure (bar)	Upstream sensor (V)	Downstream sensor (V)
0.0	0.504	0.511
1.0	0.996	1.00
2.0	1.803	1.805
3.0	2.711	2.719
4.0	3.497	3.503
5.0	4.424	4.427

Table 4.3: Pressures and corresponding voltages for the pressure sensors.

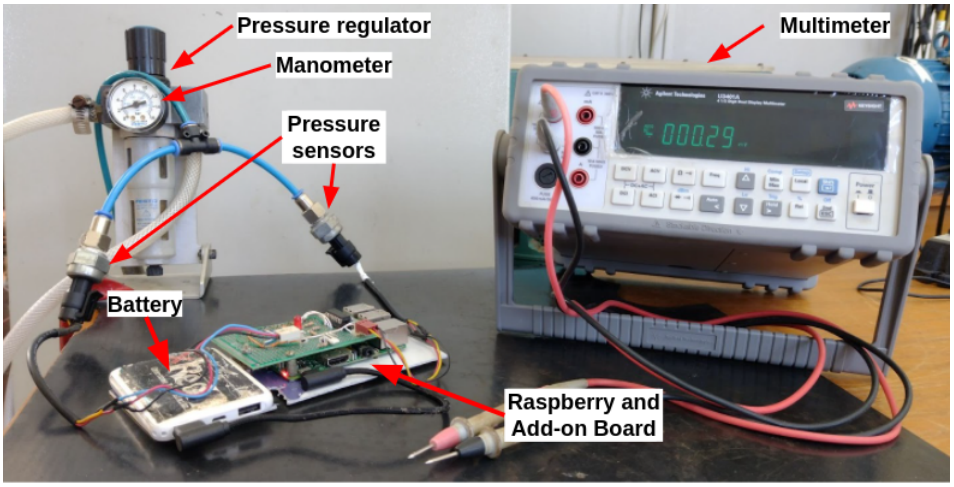


Figure 4.16: Devices used in the curve fitting procedure.

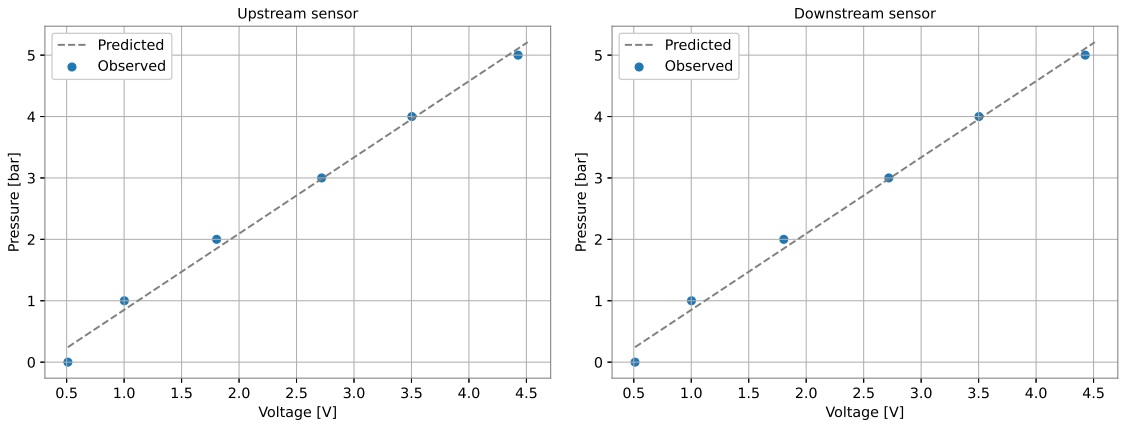
The linear least squares method was used to fit a curve for each sensor from the data presented in Table 4.3. This method defines the coefficients of a linear model that minimizes the residual sum of squares between the observed data and the data predicted by the linear model. As a result, the following expressions were found:

$$P_{UP} = 1.241 \cdot V_{UP} - 0.383 \tag{4.1}$$

$$P_{DOWN} = 1.242 \cdot V_{DOWN} - 0.390 \tag{4.2}$$

Where: P_{UP} is the upstream pressure (bar); P_{DOWN} is the downstream pressure (bar); V_{UP} is the output voltage of the upstream sensor (V); V_{DOWN} is the output voltage of the downstream sensor (V).

For each sensor, Figure 4.17 shows the observed data (Table 4.3) and the predicted data (Equations 4.1 and 4.2).



(a) Upstream sensor. (b) Downstream sensor.

Figure 4.17: Curve fitting for the pressure sensors.

4.2.3 Odometer

An odometer was attached to the PIG to measure the distance and compute the velocity of the device inside the pipeline (Figure 4.18). The odometer was built mainly with stainless steel and is composed of: (a) A base to fix the odometer on the PIG rear cover; (b) An arm that supports the wheel and the springs; (c) A wheel with a permanent magnet on its axis; (d) A Hall-effect switch (A3144), placed in order to detect the magnet installed on the wheel axis; (e) Two parallel springs to push the wheel against the duct. More details about this odometer can be found in Lima (2019).

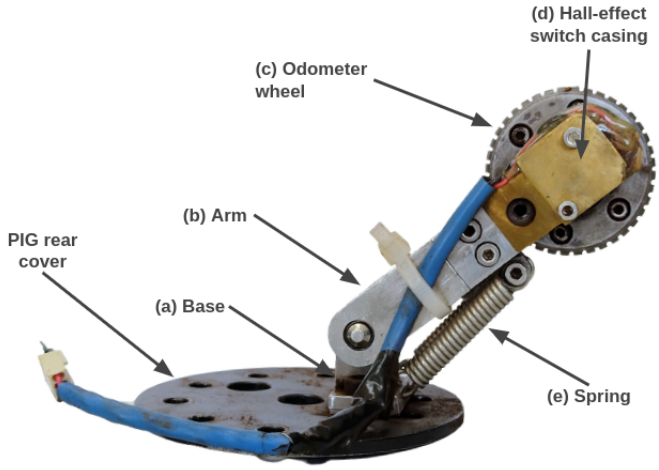


Figure 4.18: Odometer.

The Hall-effect switch allows counting the wheel revolutions by detecting the magnet attached to the wheel axis. The output of the switch goes low (0 V) when the magnetic field exceeds a certain threshold (the magnet approaches the switch); it goes high (5 V) when the magnetic field is reduced below the threshold (the magnet moves away). Therefore the output behavior of the switch is a square wave, as observed on the oscilloscope's display in Figure 4.19. To count the wheel revolutions, the Hall-effect switch was connected to a digital pin of the Raspberry, which generates an interruption in the rising edge of the square wave.

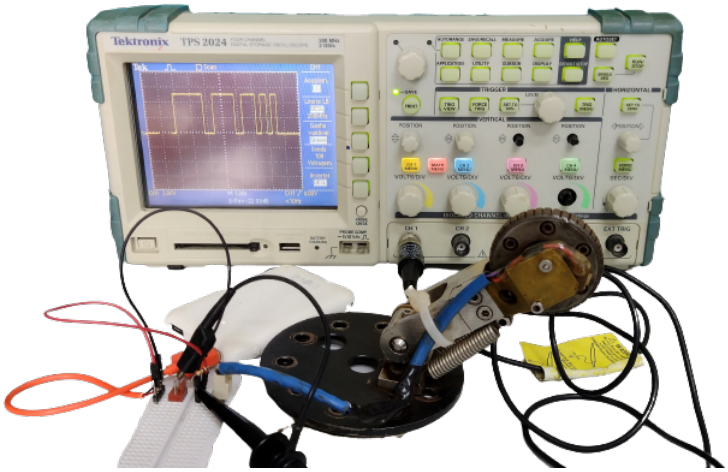


Figure 4.19: Experimental setup that illustrates the Hall-effect switch's output.

Considering that the odometer wheel has a perimeter of 0,1539 meters, the following expression was used to measure the distance:

$$x[t] = 0,1539 \cdot n[t] \tag{4.3}$$

Where: $x[t]$ is the distance travelled between time instants t and $t-T$ (m); $n[t]$ is the number of wheel revolutions between t and $t-T$; T is the sampling period.

To compute the velocity as – in the discrete case, an approximation of the derivative of distance with respect to time –, a backward Euler differentiator of order one was used:

$$v[t] = \frac{x[t] - x[t-T]}{T} \tag{4.4}$$

Where: $v[t]$ is the velocity between time instants t and $t-T$ in meters per second (m/s); $x[t]$ is the distance travelled between time instants t and $t-T$; $x[t-T]$ is the distance travelled between time instants t and $t-T$; T is the sampling period.

Velocity simulation

To define the sampling period for collecting the odometer's data, we used a function generator to simulate a square wave output of the odometer that corresponds to a PIG's velocity of 50 m/s. This value is purposely higher than the maximum velocity we observed in previous experimental data, which was equal to 20 m/s. First, it was necessary to find the relationship between this velocity and the square wave's frequency. The frequency of the wave was calculated as follows:1

- Each revolution of the wheel odometer corresponds to a traveled distance of 0.1539 m, which means that one revolution per minute (RPM) corresponds to a velocity of 0.1539 meters per minute. So the relationship between the angular velocity V_{RPM},

in revolutions per minute (RPM), and the linear velocity V_{PIG}, in m/s, is given by:

$$V_{PIG} = \frac{0.1539}{60} \cdot V_{RPM} = 0.002565 \cdot V_{RPM}[m/s]$$

- Thus for a linear velocity equal to 50 m/s,

$$V_{RPM} = \frac{50}{0.002565} = 19493.1774RPM$$

- The desired wave's frequency in hertz (Hz) is equal to the revolutions per minute divided by 60. Hence the desired frequency f is

$$f = \frac{19493.1774}{60} = 324.88Hz$$

We used $f = 325.00$ Hz for our simulation, and the velocity was calculated using the backward Euler approximation with orders 1, 2, and 3. Figures 4.20, 4.21, and 4.22 show the computed velocity for each order, and the relative error.

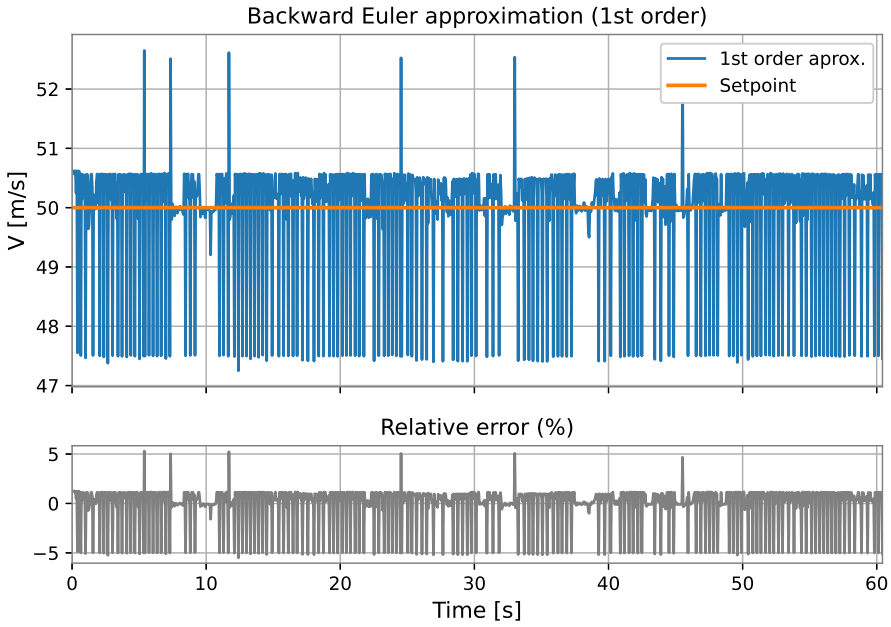


Figure 4.20: Simulated velocity using the backward Euler approximation of order 1.

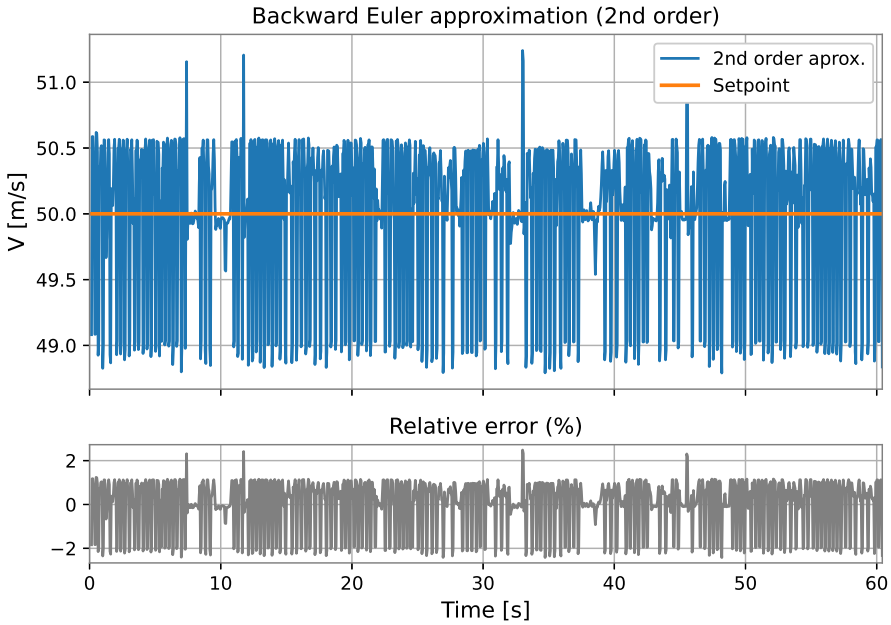


Figure 4.21: Simulated velocity using the backward Euler approximation of order 2.

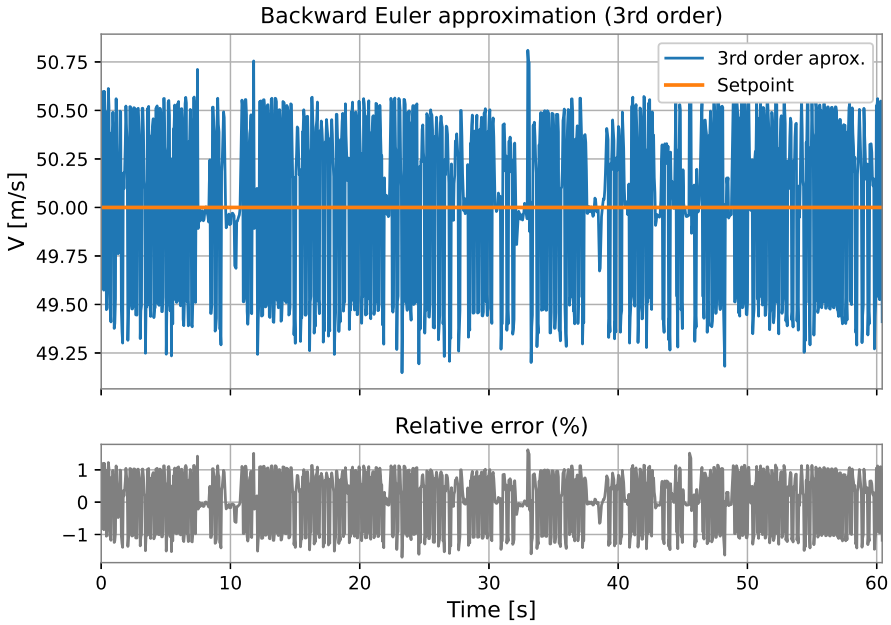


Figure 4.22: Simulated velocity using the backward Euler approximation of order 3.

In the simulations described above, the function generator was connected to a digital input of the Raspberry Pi, and a Python script recorded the number of pulses detected every 50 ms. This period is five times longer than the time spent by the Raspberry to

read the measurements from all the embedded sensors. Since the Raspberry was able to estimate the velocity correctly, the sampling period chosen was 50 ms.

4.2.4 Accelerometer

We used an accelerometer to measure the PIG's acceleration. The accelerometer is a sensor that detects accelerations by measuring the inertial forces along one, two, or three axes. It can be found in various construction types, including mechanical accelerometers, quartz accelerometers, and micro-electro-mechanical system (MEMS) accelerometers (VectorNav 2022).

A MEMS accelerometer employs a proof mass suspended to springs, which displaces in response to an external acceleration. A transducer then detects the displacement. The MPU6050 was configured to measure acceleration between -16 g and 16 g ($g = 9,8 m/s^2$). This range was chosen based on values observed in the experimental tests. Inside the PIG, the axes of the accelerometer were oriented as illustrated in Figure 4.23. The inevitable misalignment between the axes of the accelerometer and the PIG's axis of motion and the noise present in the accelerometer's output signal made it unfeasible to obtain velocity from the simple integration of acceleration.

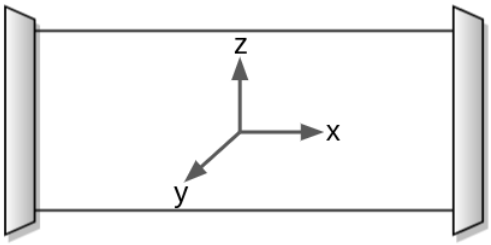


Figure 4.23: Orientation of the accelerometer inside the PIG.

We used the accelerometer MPU6050 (Figure 4.24), a MEMS device that combines a 3-axis accelerometer and a 3-axis gyroscope. The MPU6050 uses the Inter-Integrated Circuit (I²C) protocol to communicate with the Raspberry Pi.

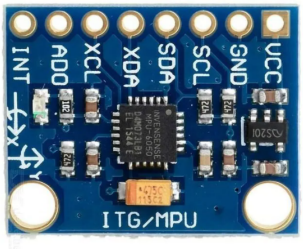


Figure 4.24: The MEMS accelerometer MPU6050 was used to measure the PIG's acceleration.

4.2.5 Pi Add-On Board

4.2.6 Power supply

A portable power bank provided the power supply for the embedded system. A USB to micro USB cable connected the power bank to the Raspberry Pi micro USB port, then the Raspberry powered the Pi Add-On Board. Table 4.4 presents the main features of the power bank.

To estimate the discharge time of the power supply, we measured the embedded system's current while simulating typical operating conditions during the PIG run, such as data collection and model inference. The maximum current consumption found for these conditions was 480 mA, which means that, for a power bank capacity of 5000 mAh, the system can work for more than 10 hours.

Feature	Description
Battery type	Lithium Polymer (LiPo)
Capacity	5000 mAh
Output voltage	5 VDC
Output current	2 A

Table 4.4: Main features of the embedded system's power bank.

4.3 PIG's testing pipeline

In order to obtain the experimental data and train the model, the testing pipeline available at the Petroleum Evaluation and Measurement Laboratory of the Federal University of Rio Grande do Norte (LAMP/UFRN) was used to perform the PIG runs.

It has an approximate length of 55 meters and a diameter between 6" and 8". Blind flanges fixed by screws were installed at the ends of the pipeline. The fluid used was compressed air, whose maximum pressure reached approximately 6 bar.

The starting point of a PIG's run is at the launcher and the endpoint is at the receiver (Figure 4.25). For launching the PIG, the launcher was pressurized up to 5 bar, and then the receiver was abruptly depressurized, causing a differential pressure that pushed the PIG along the pipeline. Finally, the run ended in the receiver, often colliding with a foam placed to absorb the impact.

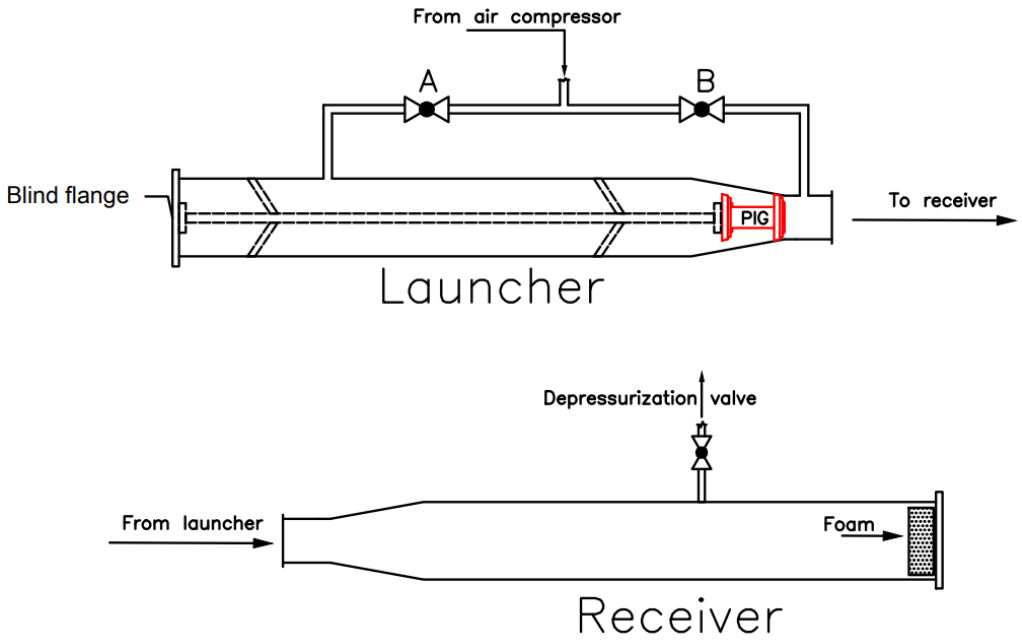


Figure 4.25: Representation of the PIG launcher and receiver. Source: modified from Freitas (2016)

Figures 4.26 and 4.27 show the top view drawing and an aerial photo of the pipeline. Further details on the development and operation of the pipeline are presented in Freitas

(2016).

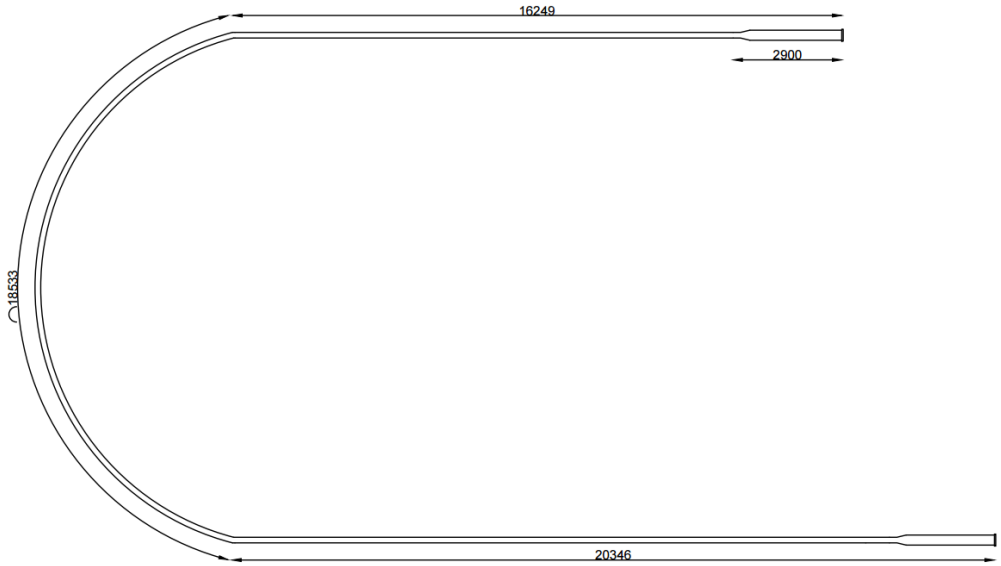


Figure 4.26: Top view drawing of the testing pipeline. Source: Freitas (2016)

Figure 4.27: Aerial photo of the testing pipeline. Source: Freitas (2016).

4.4 Software tools

This section briefly presents the main software tools and libraries employed to build the models: the Python programming language, Google Colab, Scikit-Learn, Pandas, TensorFlow, and Keras. We also used Python to write the scripts embedded on the Raspberry Pi, which are responsible for collecting and processing data from the PIG's sensors. All the libraries used in this work are free, open-source, and well documented.

4.4.1 Python

Python is a general-purpose computer programming language used to build the most diverse systems, such as websites, embedded software, and machine learning models. It is an interpreted, interactive, object-oriented language that supports multiple programming paradigms. As a result, Python has become one of the most popular programming languages (TIOBE 2022).

4.4.2 Google Colab

Google Colab is a free cloud-based tool that provides an environment to write and run Python code through the browser and is especially convenient for data science, machine learning, and education. It is based on the open-source project Jupyter Notebook (Google 2022). We used Google Colab to write and run the Python scripts related to the models' development.

4.4.3 Scikit-learn

Scikit-learn is an open-source library for machine learning in the Python programming language that David Cournapeau initially developed as a Google code summer project in 2007. It includes several machine learning algorithms, including k-means, random

forests, and support vector machines. In addition, it includes functions related to preprocessing of data and is compatible with the NumPy library (Pedregosa et al. 2011). We used Scikit-learn to preprocess the data collected from the sensors and to develop the linear regression model presented in Chapter 5.

4.4.4 Pandas

Pandas is a leading open-source library for data analysis in Python, providing highly optimized performance and several built-in functions to analyze and manipulate data. Its development began in 2008, and since 2015 Pandas has been a NumFOCUS-sponsored project (The Pandas development team 2022). We used Pandas to analyse the data collected from the PIG's sensors.

4.4.5 TensorFlow 2.0

TensorFlow 2.0 is an open-source Python library that can be used to create machine learning models developed by Google. It allows creating models directly or using libraries built on top of TensorFlow (e.g., Keras). In contrast with other numerical libraries employed in machine learning like Theano, TensorFlow was designed for research and production systems (Brownlee 2022). With TensorFlow Lite, a set of tools for on-device machine learning, it is possible to run models with optimized performance for embedded systems (reduced model size and power consumption). We used TensorFlow 2.0 indirectly, via the Keras application programming interface (API), on the construction of the machine learning models presented in Chapter 5.

4.4.6 Keras

Keras is a high-level application programming interface (API) written in Python, running on top of TensorFlow's 2.0 machine learning platform. Its focus is to enable the rapid

development of deep learning models to quickly implement simple workflows while offering a comprehensive path to advanced workflows (Chollet & others 2015). We used Keras to train most of the models presented in Chapter 5.

4.4.7 KerasTuner

KerasTuner is a scalable hyperparameter optimization library designed to be an easy-to-use tool for researchers to tune Keras models automatically. It comes with Bayesian Optimization, Hyperband, and Random Search algorithms built-in (O'Malley et al. 2019). We used KerasTuner to tune the models' hyperparameters.

4.5 Data collection

We developed a Python script to measure and record the data from the PIG's sensors as it travels inside the testing pipeline. The data were recorded to a comma-separated values (CSV) file, as shown in Figure 4.28.

```
time,num_pulses,up_pressure,down_pressure,acc_x,acc_y,acc_z
16074.288694481,3,0.15652222765021967,0.1653180263800686,0.0078125,0.85107421875,-0.46826171875
16074.399621673,4,0.15652222765021967,0.1653180263800686,0.0751953125,0.73974609375,-0.572265625
16074.510326021,4,0.16272642892037076,0.1653180263800686,-0.00048828125,0.73095703125,-0.4970703125
16074.620865113,4,0.15031802638006858,0.17152222765021968,0.08447265625,0.81298828125,-0.56884765625
```

Figure 4.28: Example of a comma-separated values (CSV) file used to record the data collected from the sensors. Source: Freitas (2016)

The column *time* is a timestamp; *num_pulses* is the number of revolutions of the odometer's wheel, which is proportional to the distance; *up_pressure* and *down_pressure* are the upstream and downstream pressures; *acc_x*, *acc_y*, and *acc_z* are the accelerations on the x, y, and z axes.

The Raspberry Pi was configured to communicate with a laptop computer using Wi-Fi and Secure Shell (SSH) protocols. This allows the user, for example, to execute commands to run the developed scripts and retrieve data without removing the embedded system from the PIG. Figure 4.29 shows the embedded system installed inside the PIG.

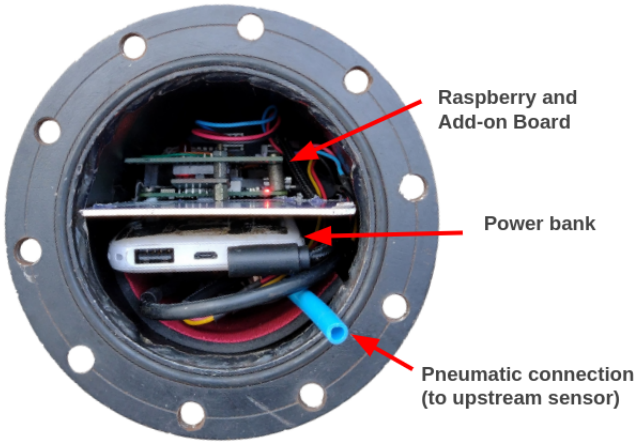


Figure 4.29: Rear view of the PIG with the embedded system installed inside. Source: Freitas (2016)

Figure 4.30 indicates the steps required for the data collection procedure. First, the electronic devices are connected to the power bank; next, the SSH connection between the laptop and Raspberry Pi is established; using the SSH client on the laptop, the command to execute the data collection script is sent to the Raspberry Pi; once the script starts running, the PIG is closed and inserted into the pipeline; finally, after the PIG is recovered from the pipeline, a command to retrieve the data (i.e., copy the CSV file) from the Raspberry Pi is sent.

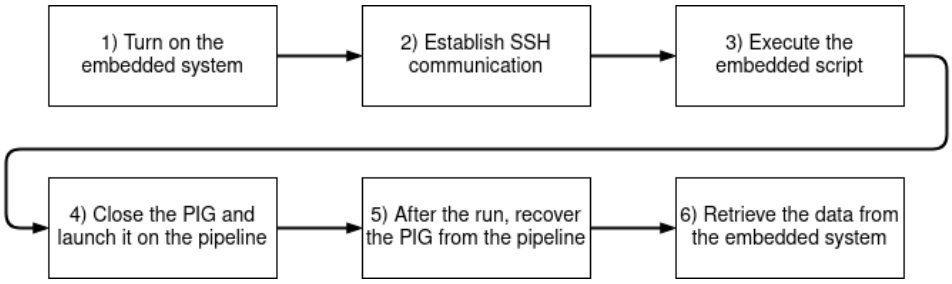
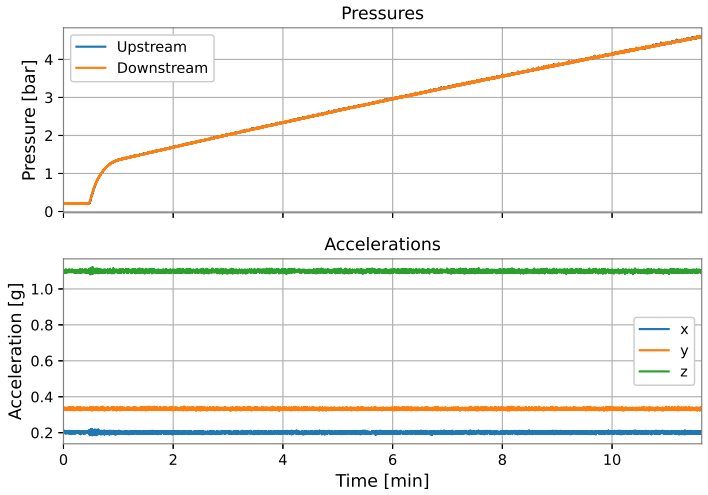


Figure 4.30: Steps of the data collection procedure.

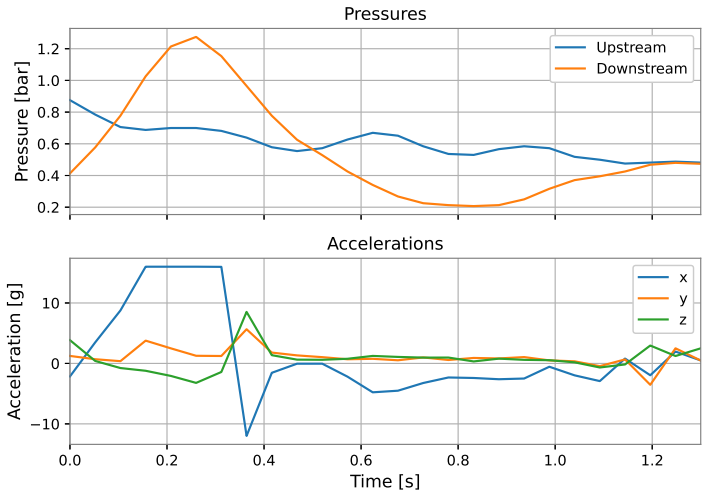
4.6 Data preparation

4.6.1 Data segmentation

After the data from a PIG's run have been retrieved, they were imported into Google Colab for data analysis and preparation. Our first step in analyzing a PIG's run was to select the region of interest for the velocity prediction model. It comprised discarding the data corresponding to (a) the pipeline's initial pressurization and (b) the PIG's collision with the receiver at the end of the duct. Figure 4.31 exemplifies these regions.



(a) Initial pressurization of the pipeline.



(b) PIG's collision at the end of the pipeline.

Figure 4.31: Examples of samples that did not belong to the interest's regions for the model's training and, hence, were discarded from the dataset.

As shown in Figure 4.31a, the upstream and downstream pressures were equal during the initial pressurization of the pipeline, before PIG's launching, and the accelerations showed no variation. Figure 4.31b shows rapid variations in the accelerations and an inversion of the pressures' signals.

4.6.2 Outliers treatment

The next step was to check for outliers, data points that differ significantly from other observations, often due to measurement errors. The outliers were replaced with the average mean of the surrounding values, as shown in Figure 4.32.

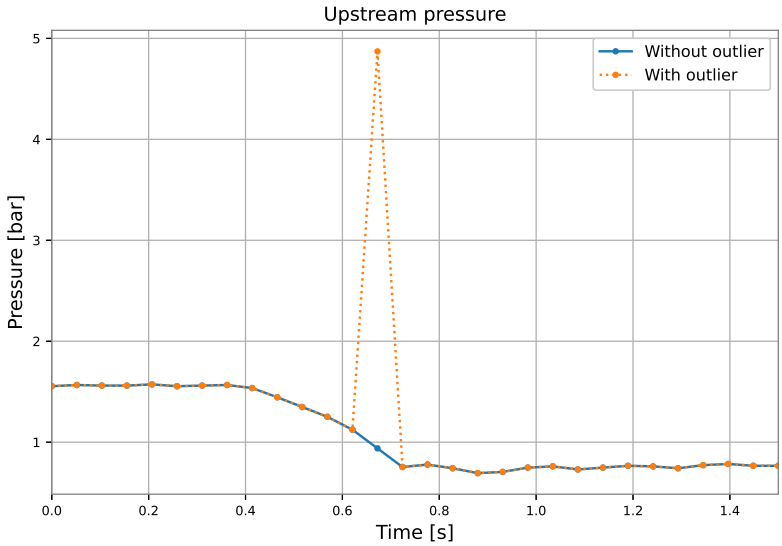


Figure 4.32: Example of pressure outlier.

4.6.3 Feature scaling

We used min-max normalization to make the features lie between 0 and 1. The general formula for min-max normalization in the range [0, 1] is given by

$$X' = \frac{X - min(X)}{max(X) - min(X)}, \tag{4.5}$$

where X' is the vector of normalized features vector and X is the vector of original features.

4.7 Chapter summary

This chapter presented the materials and methods employed in the current work. We had to develop a new prototype PIG for this work due to air leakage issues that a previous PIG (from previous research) showed. The new PIG could not avoid the air leakages, but they were significantly reduced. We presented the embedded system based on a Raspberry Pi, the sensors (pressure, velocity, and acceleration), aspects of the data collection, and the testing pipeline used for collecting the experimental data. The next chapter presents and discusses the results.

Chapter 5

Results and discussion

This chapter presents the models developed for the PIG's velocity prediction. First, we present the data sets. Next, we present the metric used to evaluate the models. Then we present the models and their performances. Finally, we discuss the results.

5.1 Data sets

We performed several runs with the PIG, but the data collected in most of the runs presented wrong values of velocity and pressure due to failures of the odometer and malfunctioning of the pressure sensors (due to air leakages into the PIG), respectively. At last, we were able to obtain viable data from two runs.

The first data set is from a run performed on 15 Nov 2021 (Run 1), comprising 310 samples and approximately 15 seconds of the run. The second data set is from a run performed on 4 Mar 2022 (Run 2), comprising 373 samples and approximately 18 seconds of the run. Both data sets were preprocessed according to the data preparation described in the previous chapter (section 4.6).

In a practical situation, first, a couple of PIG runs would be performed to obtain data to train the model, then the model would be used to predict the velocity of later runs. Aiming to represent this scenario, we used Run 1 to train the model, then Run 2 to test the model. The data collected in each run consisted of the following variables:

- The pressure upstream, i.e., behind the PIG (P_{up});
- The pressure downstream, i.e., in front of the PIG (P_{down});
- The differential pressure (ΔP), defined as

$$\Delta P = P_{up} - P_{down}; \tag{5.1}$$

- The acceleration components (acc_x, acc_y, and acc_z) measured by the 3-axes accelerometer;
- The total acceleration (acc_{total}), defined as

$$acc_{total} = \sqrt{acc_x^2 + acc_y^2 + acc_z^2}; \tag{5.2}$$

- The PIG's velocity, calculated from the odometer measurements (as explained in Chapter 4).

The variables were either measured (upstream pressure, downstream pressure, and accelerations on the three axes) or calculated (differential pressure, total acceleration, and velocity). The model's target (output variable) was the PIG's velocity, and the features (input variables) were defined for each model. Figures 5.1 and 5.2 show the training data (Run 1) and the test data (Run 2), respectively.

5.2 Model evaluation

The models were evaluated with the root mean square error (RMSE), which is given by

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}, \tag{5.3}$$

where: N is the number of samples, y_i is the true value of the i-th sample, and $\hat{y}_i$ is the predicted value of the i-th sample.

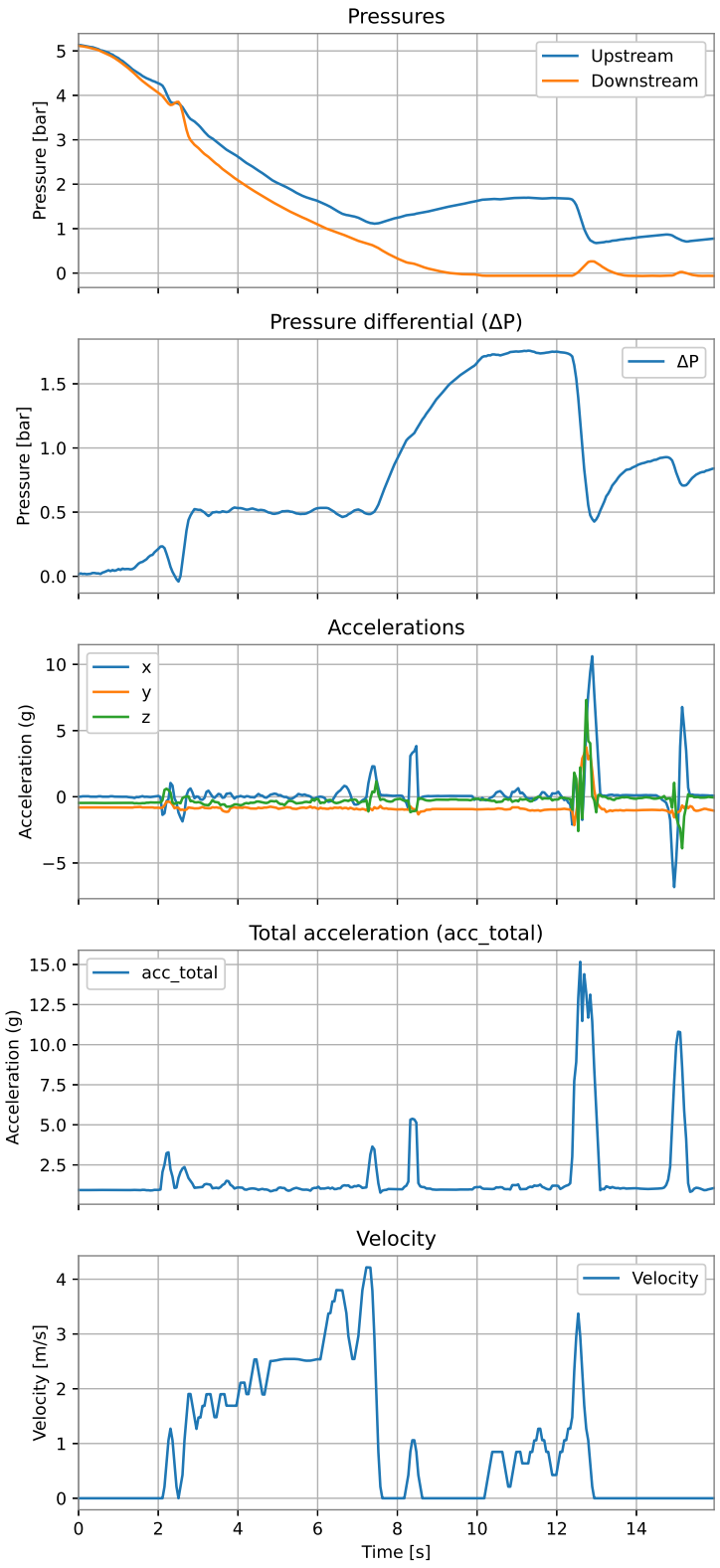


Figure 5.1: Training data set. After instant 14 s, it is possible to see a probable inconsistency in the velocity measurement, since the differential pressure and the accelerations varied significantly while the velocity remained zero.

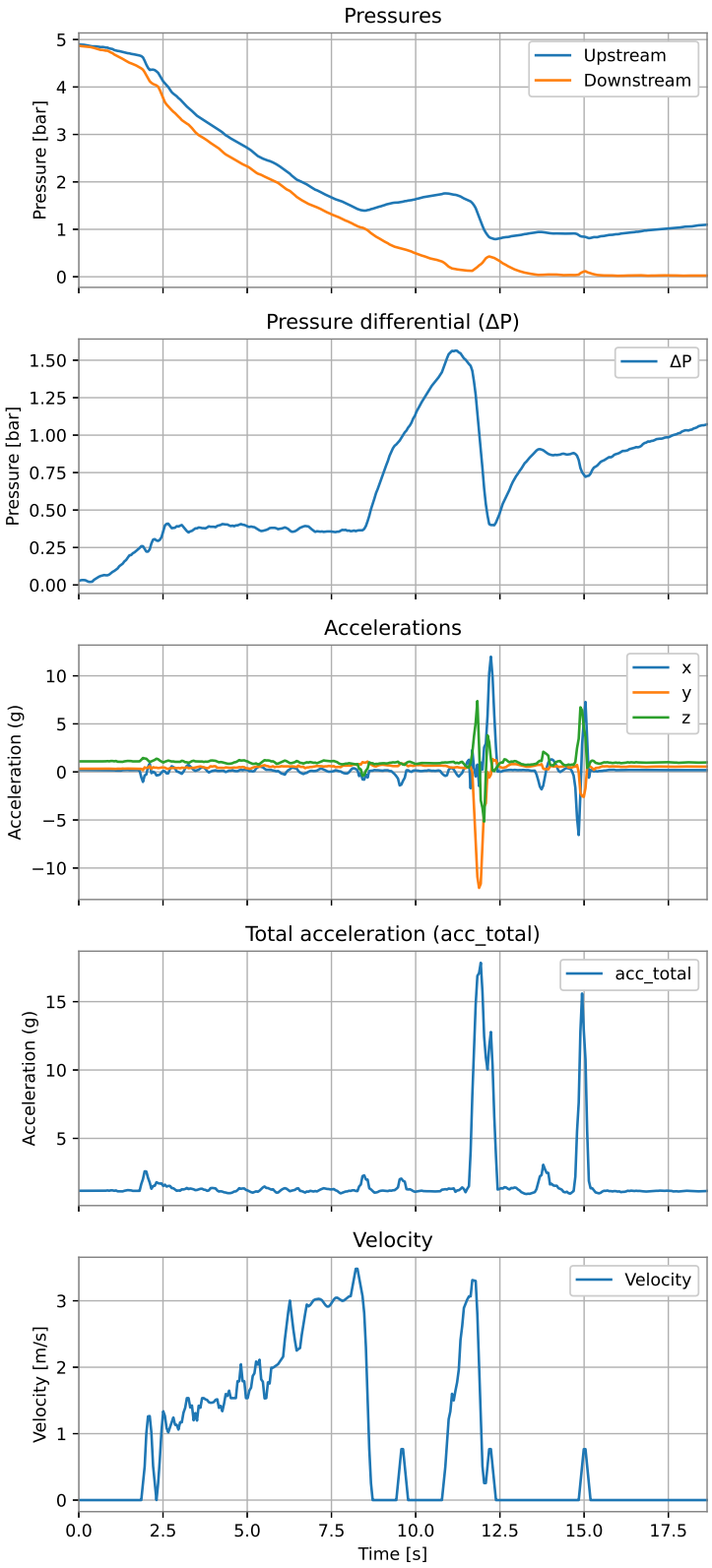


Figure 5.2: Test data set.

The RMSE measures the prediction error, determining the overall deviation between estimated and actual values. It is a widely used metric for evaluating the performance of regression models. The lower the RMSE, the better the performance of the model. We present the RMSE on the training and test sets for the models developed. The best performance on the test set was the main criterion we used to define the best model.

5.3 PIG'S velocity prediction models

Before employing more complex models to predict the PIG's velocity, we used the multivariate linear regression technique based on ordinary least-squares available in the Python library Scikit-learn (Pedregosa et al. 2011) as a baseline model. A baseline model is helpful to evaluate if a simple model like linear regression can estimate the PIG's velocity or more complex models are required (such as artificial neural networks).

First, we computed the Pearson's correlation coefficient to evaluate the linear correlation between the variables of the training data: *acc_x*, *acc_y*, and *acc_z* (accelerations on x, y, and z axes); *p_up* and *p_down* (upstream and downstream pressures); *acc_total* (total acceleration), ΔP (differential pressure), and velocity. Figures 5.3 and 5.4 show the correlations on the training and test sets, respectively, using a heat map representation.

According to the correlation heat maps, the velocity has no strong linear correlation with any of the input features, suggesting that the linear regression model might not be an adequate candidate to predict the PIG's velocity. Table 5.1 shows the training and test losses obtained by the models. Each model corresponds to a different combination of inputs.

Figure 5.5 shows the predictions of Model 3 (Table 5.1) for the training and test sets. This model presented the best performance (smaller RMSE) on the test set. The poor performance of the model confirm that it is not indicated to predict the PIG's velocity in our data sets.

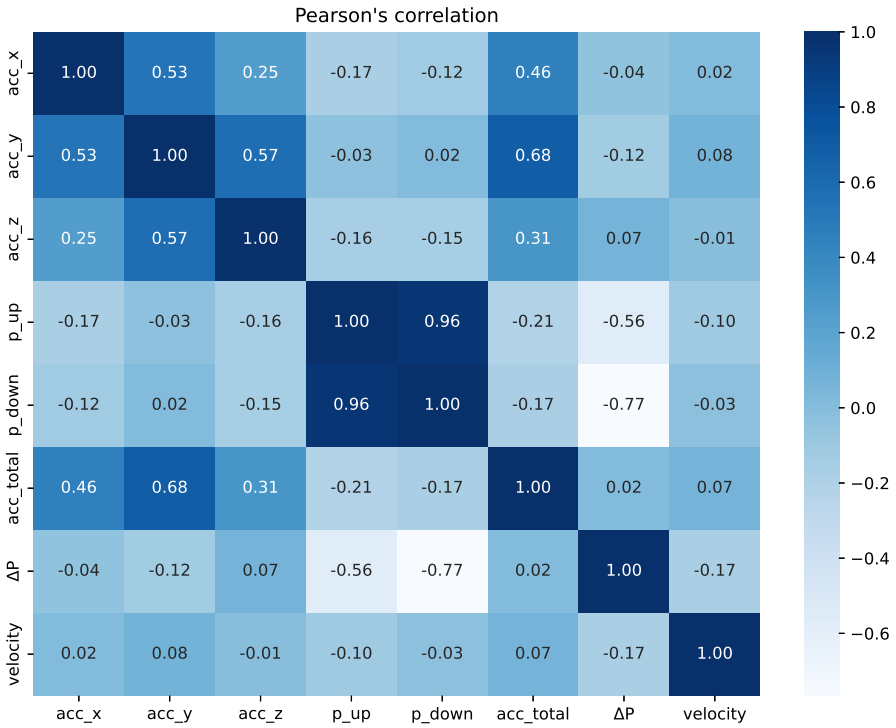


Figure 5.3: Heat map representation of Pearson's correlations for the training set.

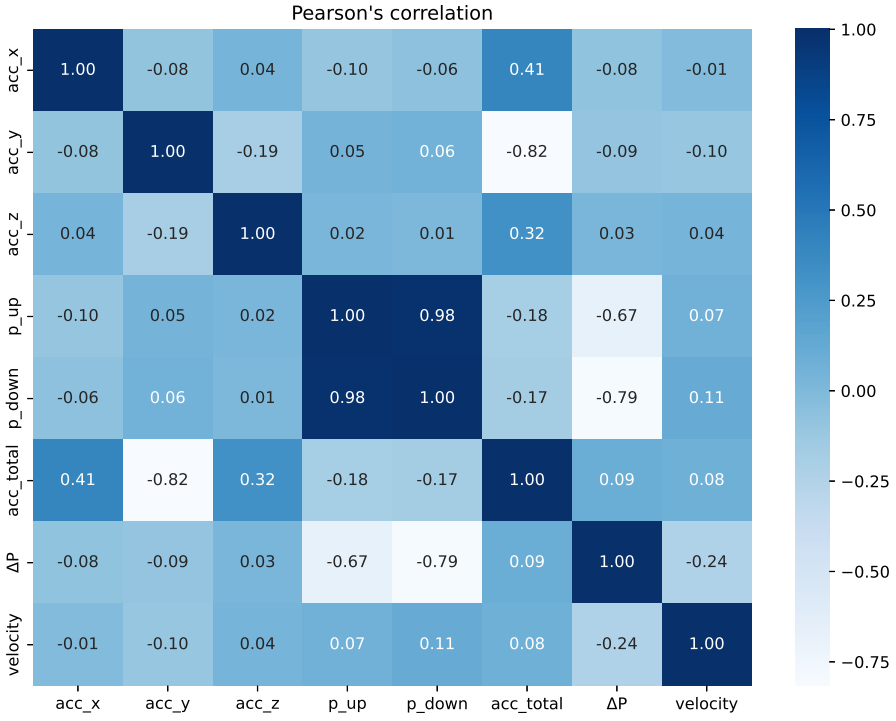


Figure 5.4: Heat map representation of Pearson's correlations for the test set.

Model	Features	RMSE (m/s)	
		Training	Test
1	All	1.1118	1.2765
2	ΔP, acc_x, acc_{total}	1.1498	1.0874
3	ΔP, acc_{total}	1.1504	1.089
4	P_{up}, P_{down}, ΔP, acc_{total}	1.1186	1.1186

Table 5.1: Performance of the linear regression models. Each model used a different combination of features. *All* means that the model used all the features from the data sets.

Next, we present different neural networks developed to predict the PIG's velocity. For all the following networks, the statements below apply:

- The parameters of the network (synaptic weights) were adjusted with the Adaptive Moment Estimation (Adam), a gradient-based optimization algorithm (Kingma & Ba 2014). The loss function was the mean squared error (MSE);
- The optimizer's learning rate and the model's hyperparameters were automatically chosen with a random search using the KerasTuner library. The search space was described for each model;
- We configured KerasTuner to randomly select 50 combinations of the hyperparameters comprised on the search space. For each combination of hyperparameters, the model was fitted three times;
- The activation function of the hidden layers was the rectified linear unit (ReLU), and the activation function of the output layer was the linear function;
- Aiming to avoid overfitting, we applied the dropout technique (rate=20%) in the hidden layers of the MLP models;
- We used a technique known as early stopping to define the number of epochs (iterations) the network was trained.

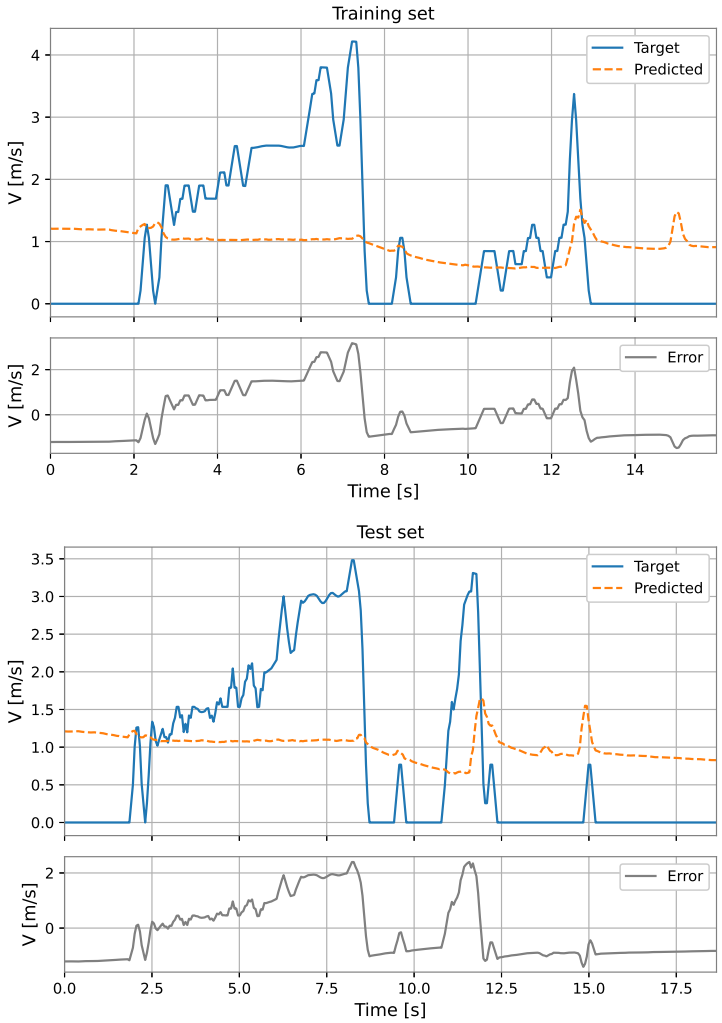


Figure 5.5: Linear regression predictions on the training and test sets. The orange dashed line is the velocity predicted by the model, the blue solid line is the target velocity, and the gray line is the absolute error, defined as target velocity minus predicted velocity.

5.3.1 Multilayer perceptron (MLP)

We built a multilayer perceptron (MLP) to predict the velocity using different combinations of the features (pressures and accelerations). The search space for the MLP model was defined as follow:

- Number of layers: $\{1, 2, 3, 4, 5\}$;
- Number of neurons in each hidden layer: $\{16, 32, 48, 64, \ldots, 256\}$;
- Learning rate: $\{0.01, 0.001, 0.0001, 0.00001\}$.

The MLP network used all the features (pressure upstream, pressure downstream, differential pressure, accelerations on the three axes, and total acceleration). The model has two hidden layers; 224 neurons in the first hidden layer and 224 neurons in the second hidden layer; learning rate equal to 0.001. The root mean square error (RMSE) on the training set was 0.2217 m/s and on the test set was 0.5457 m/s. Figure 5.6 shows the results obtained by the model on training and test sets.

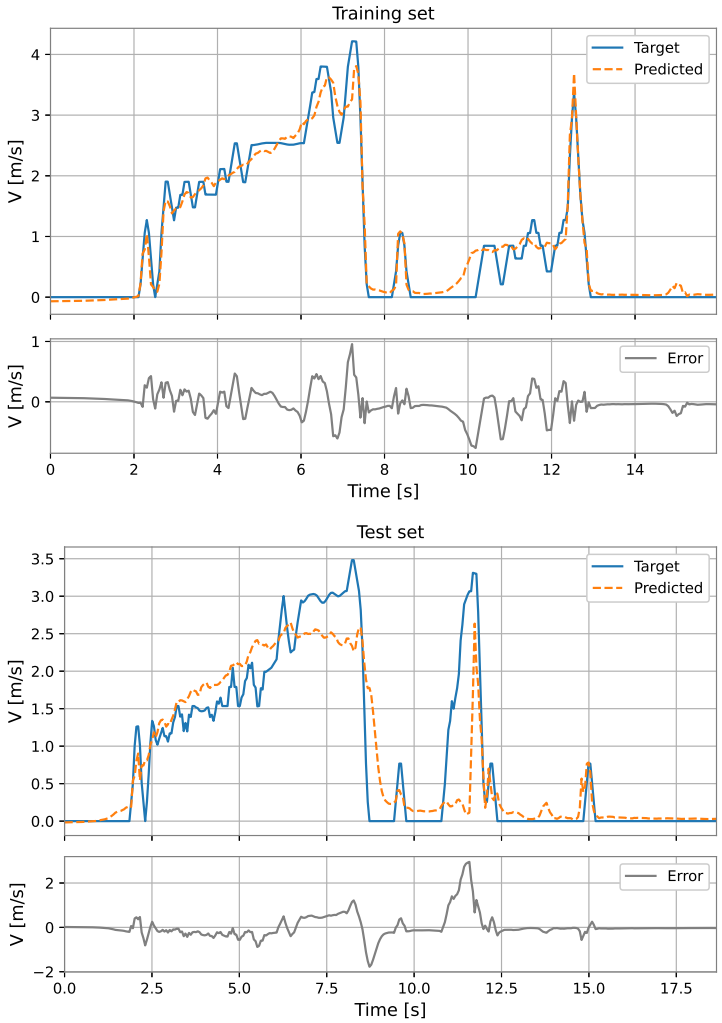


Figure 5.6: MLP's predictions on the training and test sets.

5.3.2 MLP-TDNN

The network referred to as MLP-TDNN is a time-delay neural network whose inputs are the pressures and accelerations on current instant and past instants. We tried different combinations of inputs and orders of delay (from 1 to 6). The search space for the random search was defined as follows:

- Number of layers: $\{2,3,4,5,6,7\}$;
- Number of neurons in each hidden layer: $\{16,32,48,64,\ldots,256\}$;
- Learning rate: $\{0.01,0.001,0.0001,0.00001\}$.

The MLP-TDNN model used all the features and a delay of order 1 in the inputs. The model presents 3 hidden layers; 64 neurons in the first hidden layer, 80 neurons in the second hidden layer, and 80 neurons in the third hidden layer; learning rate equal to 0.001. The RMSE on the training set was 0.2548 m/s and on the test set was 0.6091 m/s. Figure 5.6 shows the results obtained by the model on training and test sets.

5.3.3 LSTM-TDNN

Analogous to the MLP-TDNN from the last section, the LSTM-TDNN is a long short-term memory network whose inputs are the pressures and accelerations on current instant and previous instants. In this case, the architecture of the LSTM-TDNN is made up of a single LSTM layer stacked with an MLP network. The search space was defined as follows:

- Number of neurons of the LSTM network: $\{10,20,30,40,50\}$;
- Number of layers of the MLP network: $\{2,3,4,5,6,7\}$;
- Number of neurons in each hidden layer of the MLP: $\{16,32,48,64,\ldots,256\}$;
- Learning rate: $\{0.01,0.001,0.0001,0.00001\}$.

The LSTM-TDNN used all the features and a delay of order 6 in the inputs. The model presents an LSTM layer with 50 neurons stacked with an MLP network with 3

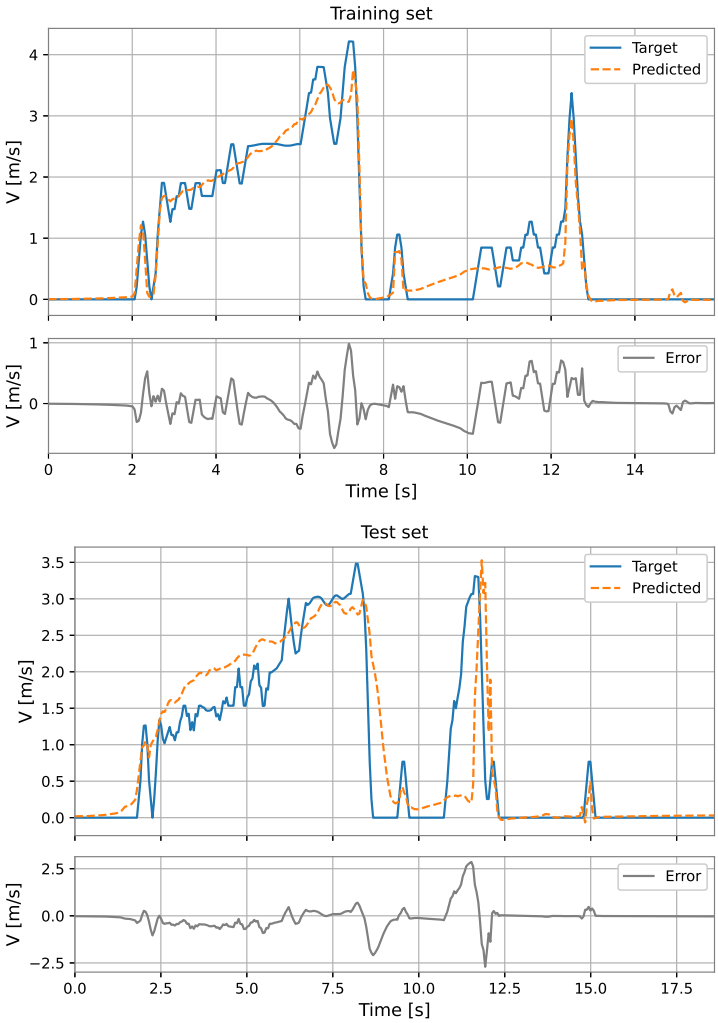
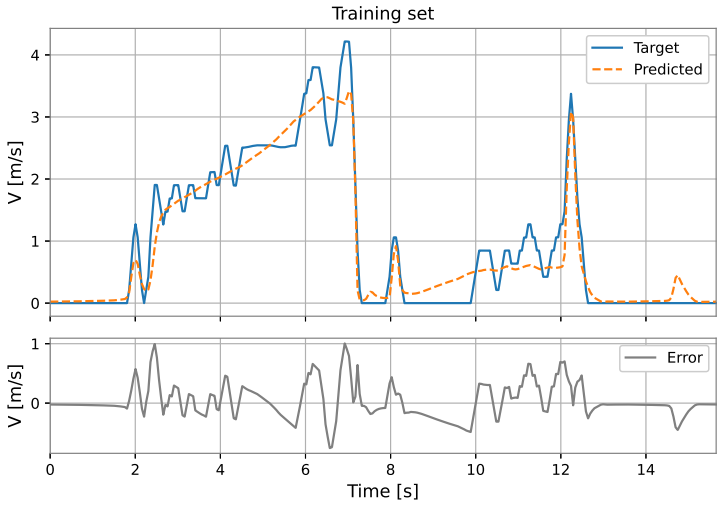


Figure 5.7: MLP-TDNN's predictions on the training and test sets.

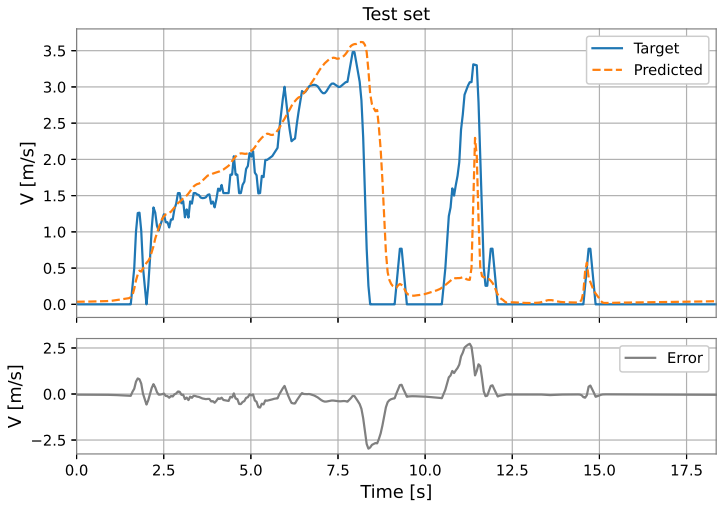
hidden layers; 224 neurons in the first hidden layer, 160 neurons in the second hidden layer, and 160 neurons in the third hidden layer; learning rate equal to 0.001. The RMSE on the training set was 0.2875 m/s and on the test set was 0.6591 m/s. Figure 5.8 shows the results obtained by the model on training and test sets.

5.3.4 MLP-NARX

The model referred to as MLP-NARX is a non-linear autoregressive network with exogenous inputs network. It refers to the series-parallel (open-loop) operation, when the model makes a one-step prediction: given the current input, the past inputs, and the past



(a) Training set.



(b) Test set.

Figure 5.8: LSTM-TDNN's predictions on the training and test sets.

true outputs, the model predicted the current output.

Again, we tried different combinations of inputs and different orders of delay (from 1 to 6). However, in this model, the delays were applied to both the inputs and to the fed-back output (velocity). The search space for the random search was defined as follows:

- Number of layers: $\{2, 3, 4, 5, 6, 7\}$;
- Number of neurons in each hidden layer: $\{16, 32, 48, 64, \ldots, 256\}$;
- Learning rate: $\{0.01, 0.001, 0.0001, 0.00001\}$.

The MLP-NARX model's features were the differential pressure and total acceleration on the current instant and past instants, besides the fed-back velocity on previous instants. The order of input delay is 1, and the order of output delay is 3. The model presents two hidden layers; 160 neurons in the first hidden layer and 192 neurons in the second hidden layer; learning rate equal to 0.001. The RMSE on the training set was 0.1314 m/s and on the test set was 0.1057 m/s. Figure 5.9 shows the results obtained by the model on training and test sets.

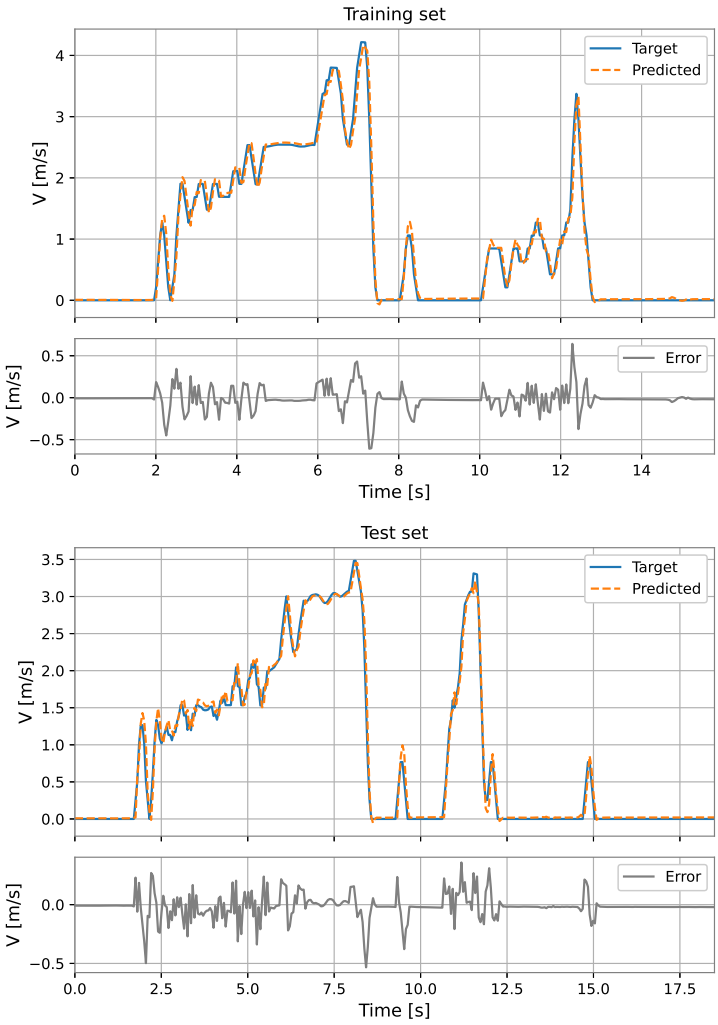


Figure 5.9: MLP-NARX's predictions on the training and test sets (series-parallel).

5.3.5 LSTM-NARX

Analogous to the MLP-NARX, the LSTM-NARX is a long short-term memory network whose inputs consist of current inputs, past inputs, and fed-back past outputs. Likewise, the LSTM-NARX refers to the series-parallel (open-loop) operation. The model's architecture is made up of a single LSTM layer stacked with an MLP network. The search space was defined as follows:

- Number of neurons of the LSTM network: $\{10, 20, 30, 40, 50\}$;
- Number of layers of the MLP network: $\{2, 3, 4, 5, 6, 7\}$;
- Number of neurons in each hidden layer of the MLP: $\{16, 32, 48, 64, \ldots, 256\}$;
- Learning rate: $\{0.01, 0.001, 0.0001, 0.00001\}$.

The LSTM-NARX model's used the differential pressure and total acceleration on the current instant and past instants, besides the fed-back velocity on previous instants. The order of input delay is 1 and of output delay is 6. The model presents an LSTM layer with 25 neurons stacked with an MLP network with 3 hidden layers; 96 neurons in the first hidden layer, 48 neurons in the second hidden layer, and 64 neurons in the third hidden layer; learning rate equal to 0.001. The RMSE on the training set was 0.2248 m/s and on the test set was 0.1780 m/s. Figure 5.10 shows the results obtained by the model on training and test sets.

5.4 Summary of results

Table 5.2 summarizes the models' performances, presenting the root mean square error (RMSE) obtained by each model on the training and test sets.

We used different architectures of neural networks and different combinations of input variables to seek for models with reasonable prediction performance. The MLP-NARX and the LSTM-NARX presented the best performances. However, it is worth mentioning

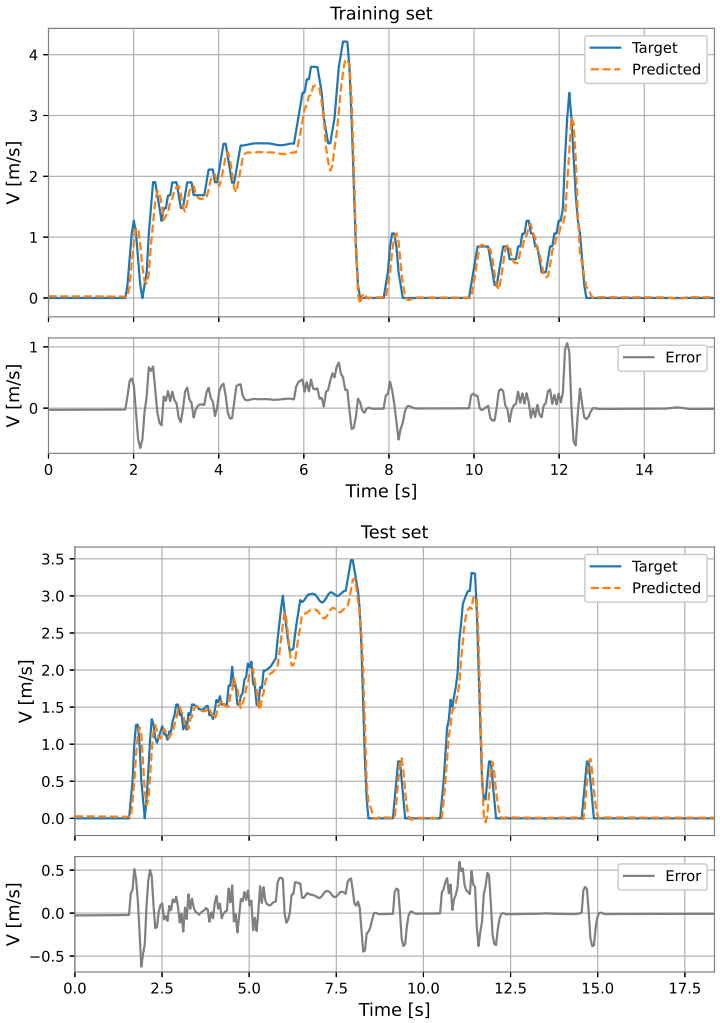


Figure 5.10: LSTM-NARX results (series-parallel).

that these results refer to the series-parallel operation of the models, requiring the true past outputs to predict the current output, while the MLP required only the current inputs. Although the MLP does not include an explicit structure for processing of temporal patterns, it achieved better performance than models like the MLP-TDNN and LSTM-TDNN in our data sets.

Model	RMSE (m/s)	
	Training	Test
MLP	0.2217	0.5457
MLP-TDNN	0.2548	0.6091
LSTM-TDNN	0.2875	0.6591
MLP-NARX	0.1314	0.1057
LSTM-NARX	0.2248	0.1780

Table 5.2: The root mean square error (RMSE) on the training and test sets obtained by each model.

5.5 Discussion

We found that artificial neural networks can predict the velocity of a pipeline inspection gauge (PIG) using the differential pressure that acts on the device. This finding agreed with its dynamical model (section 3.2), which stated that differential pressure is its driving force. In addition, the PIG's acceleration, as measured by the accelerometer, proved to enhance the performance of the networks. This was also expected since the acceleration correlates with the velocity.

We trained several neural networks with experimental data collected during two runs of a prototype PIG in a testing pipeline. We used the data collected on the first run to train the model, then used the data from the latter to evaluate it. The results show that the models developed can predict the velocity with acceptable performance even on previously unseen data (test set). Additional data are required, though, to verify the generalization capability of the models and select the best model among the already developed ones.

A simpler and widely applied technique for obtaining the velocity of PIGs is employing odometers. Their drawback is that they present significant measurement errors mainly related to the slipping and contact loss between the odometer wheel and the pipeline's wall.

A basic approach for reducing these errors is modifying constructive elements of the odometer, such as the springs and the wheel surface, altering the friction force between the wheel and the pipe wall. However, as often happens with mechanical devices, the

odometer is prone to fail (Zhu et al. 2016). Other works employed the odometer as the primary sensor but included additional information to compensate for its measurement issues. Sadovnychiy et al. (2006), for example, use the location of welds inside the pipeline; Santana et al. (2010) combined data from an odometer, a low-cost inertial measurement unit (IMU), and topographic landmarks; Zhu et al. (2016) used an IMU and the location of pipeline junctions. Finally, Araújo et al. (2018) used the differential pressure that acts on the PIG inside the pipeline to predict its velocity. These works share a common characteristic: they estimated the velocity after the PIG's retrieval from the pipeline (offline).

Similar to the research of Araújo et al. (2018), our work employed neural networks to predict the PIG's velocity. In contrast, we developed a prototype PIG with embedded sensors rather than requiring data from external pressure sensors, only available upon PIG retrieval. So our system can measure the differential pressure and, therefore, predict the PIG's velocity during its run inside the pipeline (online). The main implication of online prediction is that it allows the system to be used by a velocity controller embedded in the PIG.

Finally, we consider that our work offers a valuable contribution to the velocity measurement of pipeline inspection gauges (PIGs). Hence the oil and gas industry can benefit from our results to improve the quality of maintenance operations with PIGs, using the velocity prediction model as a complement for the odometer-based techniques.

5.6 Chapter summary

This chapter presented and discussed the results of our work. First, we showed the experimental data used to train and evaluate the models then we presented the best models we developed for predicting the PIG's velocity and the main aspects of their training. In Appendix A, we present additional models we trained while developing the models presented above.

Chapter 6

Conclusion

This work aimed to develop a model for predicting the PIG's velocity based on the differential pressure that acts on the PIG inside the pipeline. The main motivation was to provide an alternative to the velocity measurement method based on the use of odometers, considering that they present significant measurement errors caused mainly by the loss of contact between the odometer wheel and the duct surface.

According to the review presented in Chapter 2, previous studies offered improvements to the measurements of distance and velocity of PIGs based on odometers. However, previous research focused mainly on using additional sensors whose data are only available upon the PIG retrieval (offline). In contrast, the purpose of this work was to predict the velocity of the PIG during its run along the pipeline. So the system proposed in this thesis differs fundamentally from the previous works in terms of the applicability, since online measurement enables the estimator for the application of PIG velocity control.

Therefore, we expect to contribute to improve the functioning of velocity controllers for PIGs and, consequently, increase the efficiency of maintenance operations in the pipeline system.

6.1 Difficulties found

The main difficulties found in the development of this work are presented below:

- Problems in the supervisory system of the testing pipeline – The supervisory system failed to provide data that would allow us to evaluate the quality of pressure and velocity data collected by the PIG's embedded sensors inside the pipeline;
- Lack of a more reliable odometer – Several runs presented inconsistent values of distance and velocity due to odometer failures;
- PIG's sealing problems – The prototype PIG that was initially going to be used proved infeasible for the current work due to the presence of sealing problems that allowed a significant amount of air leakage into the PIG body;
- Need to build a new PIG – Building a new PIG was initially out of the scope of this work, but the sealing problems of the first prototype required us to do so;
- Restrictions due to the coronavirus disease 2019 (COVID-19) pandemic – The access restrictions related to the COVID-19 pandemic significantly reduced the time available for collecting experimental data on the testing pipeline.

6.2 Suggestions for future works

We suggest further improvements and research on the following topics:

- Install another odometer on the PIG to increase the reliability of the velocity data during the collection of training data for the models;
- Perform additional PIG runs on the testing pipeline to collect more training data for the machine learning models;
- Develop an algorithm that uses the velocity prediction model when the odometer is likely to fail (according to a fault detection method) and uses the odometer to obtain the velocity and train the model otherwise.

6.3 Related publications

The publications related to this work are presented in Table 6.1.

Journal/event	Title	Authors
Sensors 18, no. 9: 3072	Pipeline Inspection Gauge's Velocity Simulation Based on Pressure Differential Using Artificial Neural Networks	De Araújo, R.P.; De Freitas, V.C.G.; De Lima, G.F.; Salazar, A.O.; Neto, A.D.D.; Maitelli, A.L.
III CONEPETRO (2018)	Aplicação de uma rede NARX na modelagem de velocidade de um smart PIG	De Araújo, R.P.; De Freitas, V.C.G.; De Lima, G.F.; Salazar, A.O.; Neto, A.D.D.; Maitelli, A.L.

Table 6.1: Related publications until now.

Bibliography

Abiodun, Oludare Isaac, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed & Humaira Arshad (2018), 'State-of-the-art in artificial neural network applications: A survey', *Heliyon* **4**(11).

Aguirre, Luis (2007), *Enciclopédia de automática: controle e automação, volume III*, Blucher.

Aguirre, Luis Antonio (2015), *Introdução à identificação de sistemas*, 4º ed., Editora UFMG, Belo Horizonte.

Araujo, Renan Pires de (2017), Modelagem da velocidade de um pig instrumentado usando redes neurais artificiais, Dissertação de mestrado, Universidade Federal do Rio Grande do Norte, Natal.

Araújo, Renan Pires de, Victor Carvalho Galvão de Freitas, Gustavo Fernandes de Lima, Andrés Ortiz Salazar, Adrião Duarte Dória Neto & André Laurindo Maitelli (2018), 'Pipeline inspection gauge's velocity simulation based on pressure differential using artificial neural networks', *Sensors* **18**(9).
URL: *https://www.mdpi.com/1424-8220/18/9/3072*

Barnatt, Christopher (2019), 'Single board computers', `https://www.explainingcomputers.com/sbc.html`. Accessed: 06 Sep. 2019.

Batching Pigs (2019), `http://www.tdwilliamson.com/solutions/pipeline-pigging/pipeline-pigs/batching-pigs`. Accessed: 26 Jul. 2019.

Bergstra, James & Yoshua Bengio (2012), 'Random search for hyper-parameter optimization', *Journal of Machine Learning Research* **13**(10), 281–305.
URL: *http://jmlr.org/papers/v13/bergstra12a.html*

Brownlee, Jason (2022), 'Introduction to the python deep learning library tensorflow'.
URL: *https://machinelearningmastery.com/introduction-python-deep-learning-library-tensorflow/*

Chollet, François et al. (2015), 'Keras', `https://keras.io`.

Chollet, Francois (2017), *Deep Learning with Python*, 1stº ed., Manning Publications Co., USA.

Ferrari, S. & Vincenzo Piuri (2003), Introduction to neural networks for instrumentation, measurement, and industrial applications, *in* S.Ablameyko, L.Goras, M.Gori & V.Piuri, eds., 'Neural Networks in Intelligent Sensors and Measurement Systems for Industrial Applications', NATO science series. 3, IOS Press, pp. 19–42.

Fortuna, Luigi, Salvatore Graziani, Alessandro Rizzo & Maria Gabriella Xibilia (2006), *Soft Sensors for Monitoring and Control of Industrial Processes (Advances in Industrial Control)*, Springer-Verlag, Berlin, Heidelberg.

Freitas, Victor C. G., Gustavo F. Lima, Andrés O. Salazar & André L. Maitelli (2016), '"PIG" detection with pressure transducers', *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering* **5**(9), 7497–7503.

Freitas, Victor Carvalho Galvão de (2016), Desenvolvimento de sistema de supervisão para um duto de testes de pigs, Dissertação de mestrado, Universidade Federal do Rio Grande do Norte, Natal.

Freitas, Victor Carvalho Galvão de, Gustavo Fernandes de Lima, Ralyson Rayala Gonçalves de Oliveira, Andrés Ortiz Salazar, André Laurindo Maitelli & Fran-

cisco de Assis Oliveira Fontes (2014), 'Plataforma arduino no controle de velocidade de pigs', *Simpósio de Automação e Sistemas* .

Goodfellow, Ian, Yoshua Bengio & Aaron Courville (2016), *Deep Learning*, MIT Press. `http://www.deeplearningbook.org`.

Google (2022), 'Google colaboratory - frequently asked questions'.
URL: *`https://research.google.com/colaboratory/faq.html`*

Habtom, R. (1998), Soft-sensing using recurrent neural networks, *in* 'Proceedings of the 1998 IEEE International Symposium on Intelligent Control (ISIC) held jointly with IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA) Intelligent Systems and Semiotics (ISAS) (Cat. No.98CH36262)', IEEE, pp. 342–347.
URL: *https://doi.org/10.1109/isic.1998.713685*

Haniffa, Mohamad Azmi & Fakhruldin Mohd Hashim (2012), Recent developments in speed control system of pipeline pigs for deepwater pipeline applications, *in* 'World Academy of Science, Engineering and Technology', pp. 360 – 363.

Haykin, Simon (2001), *Redes Neurais: Princípios e Práticas*, 2º ed., Bookman, Porto Alegre.

Hochreiter, Sepp & Jürgen Schmidhuber (1997), 'Long Short-Term Memory', *Neural Computation* **9**(8), 1735–1780.
URL: *https://doi.org/10.1162/neco.1997.9.8.1735*

Kingma, Diederik & Jimmy Ba (2014), 'Adam: A method for stochastic optimization', *International Conference on Learning Representations* .

Liang, Zheng, Honggang He & Weili Cai (2017), 'Speed simulation of bypass hole pig with a brake unit in liquid pipe', *Journal of Natural Gas Science and Engineering*

42, 40–47.

URL: *https://www.sciencedirect.com/science/article/pii/S1875510017301178*

Lima, Gustavo F., Victor C. G. Freitas, Renan P. Araújo, André L. Maitelli & Andrés O. Salazar (2017), 'Pig's speed estimated with pressure transducers and hall effect sensor: An industrial application of sensors to validate a testing laboratory', *Sensors* **17**(9).

URL: *https://www.mdpi.com/1424-8220/17/9/2119*

Lima, Gustavo Fernandes de (2019), Proposta de controle de velocidade para ferramenta de inspeção de duto (PIG) utilizando válvula de desvio, Tese de doutorado, Universidade Federal do Rio Grande do Norte, Natal.

Lima, Gustavo Fernandes de, Victor Carvalho Galvão de Freitas, Antônio Eduardo de M. Silva, Andrés Ortiz Salazar, André Laurindo Maitelli & Francisco de Assis Oliveira Fontes (2015), Proposta de tecnologia para o controle de velocidade de pigs inteligentes utilizando lógica fuzzy, *in* 'Anais on-line do SBAI 2015', Natal, RN, pp. 1979–1984.

Liu, Shucong, Dezhi Zheng & Rui Li (2019), 'Compensation method for pipeline centerline measurement of in-line inspection during odometer slips based on multi-sensor fusion and lstm network', *Sensors* **19**(17).

URL: *https://www.mdpi.com/1424-8220/19/17/3740*

Mitchell, T.M. (1997), *Machine Learning*, McGraw-Hill International Editions, McGraw-Hill.

URL: *https://books.google.com.br/books?id=EoYBngEACAAJ*

Money, Nigel, David Cockfield, Steve Mayo & Gary Smith (2012), 'Dynamic speed control in high velocity pipelines', *Pipeline & Gas Journal* . Accessed: 03 Oct. 2019.

Narendra, Kumpati S. & Kannan Parthasarathy (1990), 'Identification and control of dynamic systems using neural networks', *IEEE Transactions on Neural Networks* **1**(1), 4–27.

Nguyen, Tan Tien, Hui Ryong Yoo, Yong Woo Rho & Sang Bong Kim (2001b), Speed control of pig using bypass flow in natural gas pipeline, *in* 'ISIE 2001. 2001 IEEE International Symposium on Industrial Electronics Proceedings (Cat. No.01TH8570)', Vol. 2, pp. 863–868 vol.2.

Nguyen, Tan Tien, Sang Bong Kim, Hui Ryong Yoo & Yong Woo Rho (2001a), 'Modeling and simulation for pig flow control in natural gas pipeline', *KSME International Journal* **15**(8), 1165–1173.

Nieckele, A. O., A. M. B. Braga & L. F. A. Azevedo (2001), 'Transient pig motion through gas and liquid pipelines', *Journal of Energy Resources Technology* **123**(4), 260–269. **URL:** *https://doi.org/10.1115/1.1413466*

O'Malley, Tom, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi et al. (2019), 'Kerastuner', `https://github.com/keras-team/keras-tuner`.

Pajankar, Ashwin (2017), *Raspberry Pi Supercomputing and Scientific Programming: MPI4PY, NumPy, and SciPy for Enthusiasts*, 1º ed., Apress, Berkeley, CA, USA.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot & E. Duchesnay (2011), 'Scikit-learn: Machine learning in Python', *Journal of Machine Learning Research* **12**, 2825–2830.

Pipeway International (2008), `https://www.pipeway.com/_files/ugd/eb138f_f12c76eee3c64f82b4be31d31ce9c1a3.pdf`. Accessed: 10 Jan. 2020.

Russell, David (2005), 'Pigging in pipeline pre-commissioning'.
URL: *http://www.ppsa-online.com/papers/2005-Aberdeen-2-Russell.pdf*

Sadovnychiy, S. & J. Lopez (2005), Improvement of pipeline odometer system accuracy, *in* 'Canadian International Petroleum Conference', Petroleum Society of Canada.
URL: *https://doi.org/10.2118/2005-013*

Sadovnychiy, Sergiy, Juan López, Volodymyr Ponomaryov & Andriy Sadovnychyy (2006), 'Evaluation of distance measurement accuracy by odometer for pipelines pigs', *Journal of the Japan Petroleum Institute* **49**(1), 38–42.
URL: *https://doi.org/10.1627/jpi.49.38*

Sahli, Hussein & Naser El-Sheimy (2016), 'A novel method to enhance pipeline trajectory determination using pipeline junctions', *Sensors* **16**(4), 567.
URL: *https://doi.org/10.3390/s16040567*

Santana, Douglas Daniel Sampaio, Newton Maruyama & Celso Massatoshi Furukawa (2010), Estimation of trajectories of pipeline PIGs using inertial measurements and non linear sensor fusion, *in* '2010 9th IEEE/IAS International Conference on Industry Applications - INDUSCON 2010', IEEE.
URL: *https://doi.org/10.1109/induscon.2010.5739911*

Silva, Ivan Nunes da, Danilo Hernane Spatti & Rogério Andrade Flauzino (2010), *Redes Neurais Artificiais para engenharia e ciências aplicadas*, Artliber, São Paulo.

Sjöberg, J., H. Hjalmarsson & L. Ljung (1994), 'Neural networks in system identification'.

Sjöberg, Jonas (1995), Non-Linear System Identification with Neural Networks, Tese de doutorado, Linköping UniversityLinköping University, Automatic Control, The Institute of Technology.

Studio, Seeed (2021), 'Water pressure sensor', https://www.seeedstudio.com/Water-Pressure-Sensor-G1-4-1-2MPa-p-2887.html. Accessed: 12 Dec. 2021.

Tangirala, A.K. (2015), *Principles of System Identification: Theory and Practice*, CRC Press, Boca Raton, FL.

The Pandas development team (2022), 'pandas-dev/pandas: Pandas 1.4.2'.
URL: *https://doi.org/10.5281/zenodo.6408044*

TIOBE (2022), 'Tiobe index for june 2022'.
URL: *https://www.tiobe.com/tiobe-index/*

Tolmasquim, Sueli Tiomno (2004), Projeto e controle da operação de passagem de pigs em dutos, Dissertação de mestrado, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

VectorNav (2022), 'Learn about mems accelerometers, gyroscopes, and magnetometers'.
URL: *https://www.vectornav.com/resources/inertial-navigation-primer/theory-of-operation/theory-mems*

Yardi, Chaitanya (2004), Design of regulated velocity flow assurance device for the petroleum industry, Dissertação de mestrado, Office of Graduate Studies of Texas A&M University, Texas.

Zhu, Xiaoxiao, Chunhua Zhao, Xiaolong Li, Shimin Zhang & Shuhai Liu (2019), 'Direct observation of odometer trajectory when passing over weld in oil and gas pipeline', *Journal of Pipeline Systems Engineering and Practice* **10**(1), 05018006.

Zhu, Xiaoxiao, Xiaolong Li, Chunhua Zhao, Shimin Zhang & Shuhai Liu (2016), 'Dynamic simulation and experimental research on the motion of odometer passing over the weld', *Journal of Natural Gas Science and Engineering* **30**, 205 – 212.

Appendix A

Additional model results

In Chapter 5 (Results and discussion), we presented the models with the best results, i.e., the models that obtained the smaller root mean square error (RMSE). In addition, this appendix includes several other models we built during this work's development, using the same data sets and methodology previously presented. The results are presented in tabular representation. The "LR" on the tables stands for the learning rate of the optimizer, "xlag" is the order of input delays, and "ylag" is the order of output delays.

We represent the neurons of the hidden layers of an MLP according to the following notation: *10:20:30* represents a feedforward model with 3 hidden layers, with 10 neurons in the first layer, 20 neurons in the second hidden layer, and 30 layers in the third hidden layer. In the case of the networks that have an LSTM layer stacked with an MLP, the number of neurons in the LSTM layers is separated by an hyphen from the hidden neurons of the MLP, as in the following example: *10-20:30:40* represents a LSTM layers with 10 neurons stacked with an MLP that has 3 hidden layers, with 10, 20, and 30 neurons, respectively.

Model	Features	Topology (hidden layers)	LR	RMSE (m/s)	
				Training	Test
1a	All	176:144:192	0.001	0.2193	0.6040
1b		256:160:160	0.001	0.2042	0.5694
1c		224:224	0.001	0.2217	0.5457
2a	ΔP, acc_{total}	64:160:176	0.001	0.4991	0.7549
2b		240:112:240:112:64	0.001	0.5185	0.8136
2c		192:16:60	0.001	0.4953	0.7561
3a	P_{up}, P_{down} ΔP	144:48:192	0.001	0.4338	0.6523
3b		240:96:64:240	0.001	0.4386	0.6754
3c		256:144	0.001	0.4368	0.6711

Table A.1: MLP model results.

Model	Lag	Features	Topology (hidden layers)	LR	RMSE (m/s)	
					Training	Test
1a	xlag=1	All	224:160:208:192	0.001	0.2439	0.7211
1b			64:80:80	0.001	0.2548	0.6091
1c			96:144:208:192:48:128:144	0.001	0.2553	0.7647
2a	xlag=3	All	48:112:80:80:224	0.001	0.2414	0.7903
2b			160:128:224:64:144:48	0.001	0.2179	0.6597
2c			160:64:240:112	0.001	0.2432	0.7353
3a	xlag=6	All	192:224:48:240	0.001	0.2029	0.7465
3b			128:32:240:128	0.001	0.2306	0.7412
3c			240:128:256:208	0.001	0.2159	0.8938
4a	xlag=1	ΔP, acc_{total}	128:128:160:240	0.001	0.4365	0.7472
4b			224:64:240:16	0.001	0.4394	0.8139
4c			208:144:32	0.001	0.4452	0.7584
5a	xlag=3	ΔP, acc_{total}	80:208:144	0.001	0.3754	0.6804
5b			176:96:80:64:176	0.001	0.3898	0.7277
5c			96:192:160:80:256:160	0.001	0.3956	0.691
6b	xlag=6	ΔP, acc_{total}	192:208:96	0.001	0.3992	0.6640
6c			80:208:208:208:208:112	0.001	0.3876	0.6533
			224:144:208:64:96	0.001	0.4042	0.6789

Table A.2: MLP-TDNN model results.

Model	Lag	Features	Topology (lstm-hidden)	LR	RMSE (m/s)	
					Training	Test
1a			45-160:160:256	0.001	0.3269	0.8036
1b	xlag=1	All	20-128:256:48:176	0.001	0.3345	0.7001
1c			35-176:144	0.001	0.3932	0.8271
2a			25-112:96:208:208:128	0.001	0.3484	0.7501
2b	xlag=3	All	45-224:192	0.001	0.3731	0.9315
2c			45-112:240:240:128	0.001	0.2990	0.7083
3a			40-96:128:64:128	0.001	0.2782	0.7728
3b	xlag=6	All	50-224:160:160	0.001	0.2875	0.6591
3c			10-176:80	0.001	0.3393	0.9451
4a			25-48:128:240	0.001	0.7165	0.9016
4b	xlag=1	ΔP	40-64:192:192	0.001	0.7042	0.8538
4c			40-160:32:112:160	0.001	0.7500	0.8735
5a			20-144:224:144:128	0.001	0.7247	0.8397
5b	xlag=3	ΔP	20-192:96	0.001	0.7325	0.8652
5c			40-240:112:64:160	0.0001	0.7422	0.8767
6a			30-128:160:160:224	0.001	0.7111	0.8581
6b	xlag=6	ΔP	20-144:128:48:160	0.001	0.6965	0.8321
6c			35-64:224	0.001	0.6991	0.8610
7a			25-256:96	0.001	0.6075	0.8265
7b	xlag=1	ΔP, acc_{total}	35-144:208:96:208	0.0001	0.7277	0.8902
7c			35-32:96:16	0.001	0.6555	0.8701
8a			40-96	0.001	0.5809	0.8238
8b	xlag=3	ΔP, acc_{total}	15-240:256:16	0.001	0.5166	0.7935
8c			50-160:112:48	0.001	0.5686	0.7741
9a			45-240:256	0.001	0.4735	0.8052
9b	xlag=6	ΔP, acc_{total}	10-144:112:208:144:16	0.001	0.4498	0.7478
9c			15-224:32:240	0.001	0.4555	0.7247

Table A.3: LSTM-TDNN model results.

Model	Lag	Features	Topology (hidden)	LR	RMSE (m/s)	
					Training	Test
1a			208:160:96	0.001	0.1593	0.2717
1b	xlag=1,ylag=1	All	208:32:69:160:192:48	0.001	0.3048	0.4019
1c			176:160:64:128	0.0001	0.1950	0.2123
2a			32:208:64	0.001	0.2225	0.2460
2b	xlag=1,ylag=3	All	80:240:160:192	0.001	0.2312	0.2596
2c			48:64:64:240	0.001	0.3522	0.3833
3a			96:102	0.001	0.1253	0.1531
3b	xlag=3,ylag=1	All	128:244	0.0001	0.1724	0.1625
3c			176:128:224	0.0001	0.1708	0.1721
4a			224:240	0.0001	0.1268	0.1093
4b	xlag=3,ylag=3	All	128:240:80:192:112	0.0001	0.2970	0.3504
4c			208:144:176	0.001	0.1268	0.1394
5a			160:208	0.0001	0.1296	0.1493
5b	xlag=1,ylag=6	All	64:256:128	0.0001	0.2017	0.2008
5c			112:176:224	0.001	0.3038	0.3134
6a			240:240	0.00001	0.1996	0.1590
6b	xlag=1,ylag=1	ΔP, acc_{total}	192:160:256	0.001	0.2030	0.1753
6c			208:80	0.0001	0.1868	0.1518
7a			160:192	0.001	0.1314	0.1057
7b	xlag=1,ylag=3	ΔP, acc_{total}	176:192:256	0.0001	0.1900	0.1870
7c			208:176	0.0001	0.1397	0.1093
8a			48:224:192:208	0.001	0.2818	0.2449
8b	xlag=1,ylag=6	ΔP, acc_{total}	240:176:80:128	0.0001	0.1712	0.1450
8c			80:240:128:144:192	0.0001	0.3065	0.2809
9a			192:176	0.0001	0.1753	0.1446
9b	xlag=3,ylag=1	ΔP, acc_{total}	48:240:144:240	0.0001	0.3124	0.2826
9c			256:112:64	0.001	0.1614	0.1502

Table A.4: MLP-NARX results.

Model	Lag	Features	Topology (LSTM-Dense)	LR	RMSE (m/s)	
					Training	Test
1a			30-64:192:160:112	0.001	0.3067	0.4284
1b	xlag=1,ylag=1	All	30-176:192:192	0.001	0.3556	0.3175
1c			45-240:192	0.0001	0.3158	0.2785
2a			40-48:80	0.001	0.2542	0.2264
2b	xlag=1,ylag=3	All	10-32:128	0.001	0.2694	0.2401
2c			50-144:208:256:160	0.0001	0.3063	0.2956
3a			25-240:96:240	0.0001	0.3174	0.2750
3b	xlag=3,ylag=1	All	20-112:48:96	0.001	0.3284	0.2988
3c			45-112:160:240:64	0.001	0.4903	0.4525
4a			20-224:176	0.001	0.2647	0.2397
4b	xlag=3,ylag=3	All	10-144:256	0.001	0.2802	0.239
4c			40-64:96:128:80	0.001	0.2987	0.2643
5a			45-32:16:64	0.001	0.407	0.3706
5b	xlag=1,ylag=6	All	5-240:192:176	0.0001	0.2764	0.2516
5c			30-224:192	0.0001	0.2931	0.2563
6a			35-144	0.001	0.2921	0.2664
6b	xlag=1,ylag=1	ΔP, acc_{total}	20-32:192:128:208	0.001	0.4447	0.4117
6c			40-64:192:224:240	0.001	0.4785	0.4420
7a			20-224:224:176:240:16	0.001	0.4241	0.3769
7b	xlag=1,ylag=3	ΔP, acc_{total}	40-224:128:176:32	0.0001	0.3123	0.2523
7c			15-192	0.0001	0.3451	0.2866
8a			25-96:48:64	0.001	0.328	0.2867
8b	xlag=1,ylag=6	ΔP, acc_{total}	50-80	0.001	0.2248	0.1780
8c			10-192:224:208:160	0.001	0.3589	0.3049
9a			15-192	0.0001	0.3781	0.3597
9b	xlag=3,ylag=1	ΔP, acc_{total}	25-128:32:192:96	0.001	0.2852	0.2497
9c			25-144:240	0.0001	0.3384	0.3234

Table A.5: LSTM-NARX results.

Appendix B

Electronic schematic of the Pi Add-On Board

Figure B.1 presents the electronic schematic of the Pi Add-On board.

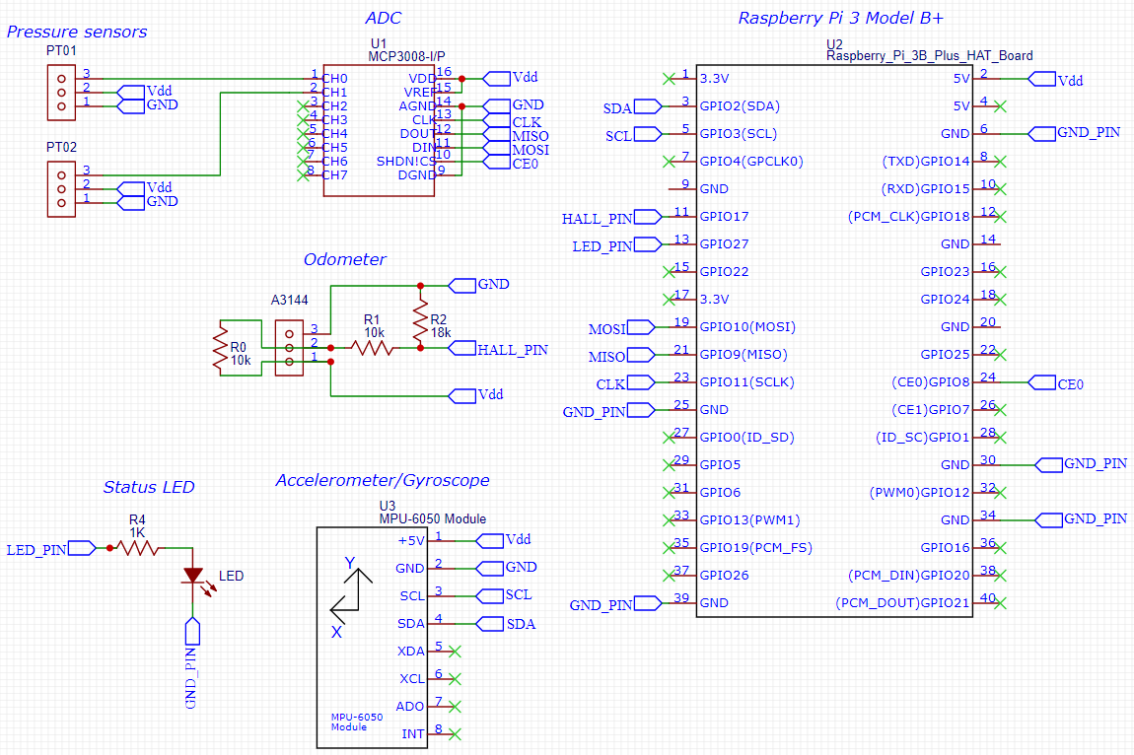


Figure B.1: Electronic schematic of the Pi Add-On Board and other hardware elements of the embedded system.