



Faculty IV

School of Science and Technology

Institute for Automation

Chair for Electrical Machines, Drives and Controls

Prof. Dr.-Ing. Schröder

Master Thesis

Name : Manoj Venkatarao Sanagapalli

Matr.-Nr. : 1383616

Fault Diagnostics for Mobile Robot Navigation

Workplace : Robert Bosch GmbH

Advisor : Dr. Ingo Lütkebohle



Submission Data : 29/06/2020

Declaration

I hereby declare that this Master thesis entitled **Fault Diagnostics for Mobile Robot Navigation**, is the result of my own work and it has not been submitted for any other degree or other purposes. Further, I have faithfully and accurately cited all sources that assisted me throughout my Master's thesis work, including other researchers' work published or unpublished.

I am aware of the consequences of not recognizing any sources used in the respective work and it is considered as fraud and plagiarism.

.....

Manoj Venkatarao Sanagapalli

Matr.-nr.: 1383616

June 2020

Acknowledgement

I would like to thank my advisor in Robert Bosch, Dr. Ingo Luetkebohle for his valued support and guidance throughout my thesis work. I would like to thank all my colleagues of SW Platforms and Technologies department for their motivation making my stay pleasant and enjoyable.

I gratefully acknowledge my supervisor at the University of Siegen Prof. Dr.-Ing. Günter Schröder for his guidance, feedback and cooperation for my thesis.

I deeply thank my parents, Purnachandar Sanagapalli and Anantha Lakshmi Sanagapalli for their constant emotional support and being my strength in every aspect of life.

Abstract

Fault Diagnostics for Mobile Robot Navigation

by

Manoj Venkatarao Sanagapalli

Matriculation Number: 1383616

Submitted to the Chair for Electrical Machines, Drives and Controls (EMAS),
Department of Electrical Engineering and Computer Science on Jun. 2020 in
Partial Fulfillment of the Requirements for the Degree of

Master of Science in Mechatronics

Robots are evolving from powerful stationary mechanical systems to sophisticated, mobile platforms to address a broad range of industrial and domestic needs. Mobile robots became the center of robotic innovations in the present world because of its widespread usage in different areas like industrial, military, domestic and consumer applications. The complex and dynamic environment may lead to different types of faults endangering the mobile robot and its surroundings. So, a mechanism to detect and identify the faults during run-time is a necessary task for a mobile robot. Thus Fault diagnosis system for mobile robotics is challenging and an area worth to study. The main purpose of this work is to design and develop a fault diagnosis mechanism for a mobile robot using run time data from a robot.

For this, the Kobuki robot is used for experimentation. The robot is controlled

using drivers provided by ROS (Robot operating system) software. Data from the robot's sensors is extracted in different faulty cases using a defined experimental approach . This extracted data is used to train a machine learning model. KNN classification model is used to train and classify the data in to different classes. The trained model is then loaded in the software of the overall system and is tested on the Kobuki robot. This developed component is able to detect faults in real time while the robot is moving.

Table of Contents

Table of Contents	i
List of Figures	iii
List of Tables	viii
1 Introduction	1
2 Related work	5
2.1 State of the art	5
2.2 Problem statement	9
3 Background	11
3.1 Machine Learning	11
3.1.1 Nomenclature	13
3.1.2 Classification problem	16
3.2 Robot Operating System	20

3.2.1	ROS Nomenclature	22
4	Design and Methodology	27
4.1	System Architecture	27
4.1.1	Kobuki Robot	27
4.1.2	Raspberry pi	29
4.2	Software Architecture	31
4.3	Implementation	34
4.3.1	Use cases	34
4.3.2	Machine Learning Implementation	39
5	Results	45
5.1	ROS Implementation	45
5.1.1	Machine learning	49
6	Discussion	57
6.0.1	Conclusion	58
6.0.2	Future work	59
	References	61
	Appendix A	65
A.1	Data streams	65

Appendix B	69
B.1 Implementation	69
B.1.1 Visualization	69
B.1.2 Machine Learning	74

List of Figures

1.1	Types of Mobile robots [7]	2
1.2	Fault tolerance diagram [3]	3
2.1	Subsystems [13]	6
2.2	Observer residual generation [5]	7
2.3	Execution monitor [15]	8
2.4	Binary class	9
2.5	Ternary class	10
3.1	Supervised Learning	12
3.2	Reinforcement Learning	13
3.3	ROC Curve [14]	16
3.4	Classification workflow	17
3.5	Linear Regression	18
3.6	Decision Tree	19
3.7	k-NN classification [1]	20

3.8	Node structure	22
3.9	Fields	23
3.10	ROS2 Architecture	25
3.11	Folder Structure	26
4.1	Kobuki Robot	28
4.2	Byte Stream Structure [4]	28
4.3	Serial pin-out [17]	29
4.4	Rasberrypi 3B	30
4.5	System Architecture	30
4.6	Hardware Architecture	31
4.7	Kobuki node	32
4.8	Navigation Node	32
4.9	Csv Extractor Node	33
4.10	Validation node	34
4.11	Ros Implementation	35
4.12	Rigid Obstacle	36
4.13	Movable Obstacle	36
4.14	Slip	36
4.15	Scratch	37

4.16 Stuck	37
4.17 Wedge	38
4.18 Wheel drop	38
4.19 Splitting	40
4.20 Training and Testing	42
4.21 Confusion Matrix	43
4.22 No. of neighbors	43
5.1 Launching nodes	45
5.2 Data extraction	46
5.3 Feature plot	46
5.4 Confusion Matrix : Manual binary classification	47
5.5 Three classes	48
5.6 Confusion Matrix : Manual binary classification	48
5.7 Dataframe object	49
5.8 Cleaning	50
5.9 Correlation	51
5.10 Data Split	51
5.11 Scaling	52
5.12 Output - Binary classification	53

5.13	Output - Three classes	54
5.14	Confusion Matrix : Multi-class	55
5.15	Classification Report : Multi-class	55
5.16	Run time predictions	56
A.1	IMU byte stream	66
A.2	Current byte stream	66
A.3	Basic Core Sensor Data	67
B.1	Launch file	69
B.2	No obstacle	70
B.3	Rigid obstacle	70
B.4	Movable obstacle	71
B.5	Stuck wheels	71
B.6	Slip	72
B.7	Slip	72
B.8	Wedge	73
B.9	Pit	73
B.10	Importing packages	74
B.11	formatting data	74
B.12	Training	74

List of Tables

4.1	Technical Specifications [17]	27
4.2	Description	39

Chapter 1

Introduction

Mobile robots are no longer a luxury in the workplace; they are a necessity [6]. In recent years, mobile robots have seen a rapid increase in their applications over diversified fields like logistics, warehousing, medical & healthcare, and consumer appliances. Mobile robots are been used in applications where it is dangerous or impossible for human interventions like mine detection and space exploration. NASA is expecting to launch a mobile robot named VIPER which is an exploration rover for its resource mapping mission on the moon by 2022. In automated warehouse storage and retrieval systems - a recent advancement in the field of logistics uses mobile robots for industrial warehouse management which include tasks like picking objects and placing them in designated storage locations. Autonomous lawn mower is one of the widely used consumer robot. Bosch Indego lawnmower is a product developed by Bosch in the field of mobile robots for household appliances.



Figure 1.1: Types of Mobile robots [7]

Faults occurring in a mobile robot is a bottleneck for the safety and reliability of the robot, especially when used in a public-oriented environment. Current research in the mobile robotics community is more focused on making them more dependable and fault-tolerant. Fault tolerance is a combination of fault diagnosis and fault recovery. When a mobile robot is moved in a complex dynamic environment, it should be able to diagnose faults and change its course of action through fault recovery.

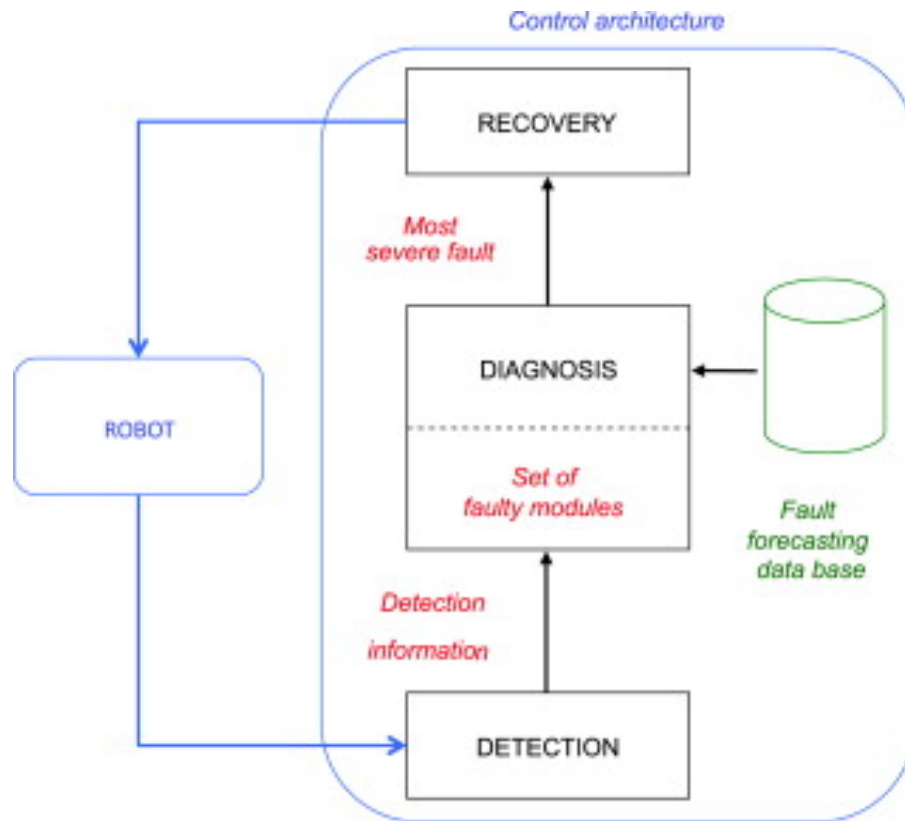


Figure 1.2: Fault tolerance diagram [3]

Even a well designed and tested robot may encounter faults [2]. The reason maybe because of the degradation of components or a misinterpretation of the environment. Faults, when not addressed may cause failures in the entire system and it is important to detect faults in a timely manner to avoid expensive consequences [18]. If failures are not detected, they may lead to dangerous behaviors of the robot and thus compromising the safety of the surroundings. Also the delay in detection of faults may lead to hardware breakdowns. So, fault diagnosis in mobile robotics is a very interesting subject to do scientific research. Mobile robots generally have fault handling as part of there hardware and software design, but handling all faulty cases manually takes lot of efforts. Hence there is a need of a fault diagnosis system that can detect the faults automatically during real time of robot operation.

Faults in a mobile robot can occur from a large space of possibilities; robot sensors can be faulty; actuators have mechanical losses; uncertainty in the environment. So,

fault detection and identification is a complex problem considering the internal systems and limited computation power of the robot. Hence, the main objective of this work is to develop a robust real time fault diagnosis system that can detect faults by analyzing the behavior of the robot during the motion of the robot. we present a statistical approach implemented on an actual robot to obtain an optimized fault diagnosis mechanism.

Chapter 1 provides the introduction and motivation behind the project undertaken.

Chapter 2 provides information about previous research that was conducted on this topic and the main hypothesis that is considered for implementation

Chapter 3 presents the background information about machine learning which is used to classify faults of the fault diagnosis mechanism. In the second part, the background information about ROS (robot operating system) is provided which is the software development environment used to operate the robot used in this project.

Chapter 4 explains the design and methodology of the project. System architecture and software architecture is described with the details of all hardware and software components used in the project. Then details about the implementation are described.

Chapter 5 presents the results obtained after implementation.

Chapter 6 concludes the project and future scope of continuation is addressed.

Chapter 2

Related work

2.1 State of the art

The main objective of this project is to diagnose faults to prevent any discrepancies in the behavior of a robot. As the applications of mobile robots are widely spread in many areas, it is very likely to have faults in the operation of mobile robots that hinder the functionality [9]. Real-time fault diagnosis increases robot efficiency by completing tasks and increasing safety and performance. The ability of an internal system to detect faults is a major asset to draw conclusions about future actions of the robot to prevent mission abortion (in case of highly technical robots), material damage, and accidents with the human. Fault-tolerant machines can be easily repaired and are more flexible to be operated in an unknown environment. There are different kinds of fault detection approaches prevailing in the modern robotics research community. [11] presents the comparison of well-known techniques that are used for fault diagnosis for mobile robots and analyzed the suitability of algorithms for particular mobile robots. The majority of the approaches are model-based fault detection and isolation which uses mathematical models for fault detection.

The mathematical model of the robot is created as an approximation of the actual

system and is given some inputs to run. When there is a fault, the output of the model deviates from the actual output and the deviation is termed as "residual". Based on the system variables and residual analysis, the faults are detected and timely warnings are provided. As the sensor noise is very common in robotics, sensitivity poses a major disadvantage to this technique. Residuals become non-zero with a small amount of noise and if the model is not exactly determined. A particle-based fault diagnosis algorithm is one of the state-of-the-art in the robotic domain. Particle filters Gustafsson(2000) process state variables of a system to obtain the status of the current state, given previous state, transition model, and current observation. Weights are assigned to particles in the system. For every time step, weights associated to set particles are updated and new set particles are replaced using transition probability. These new weights are used to compute the probability distribution of the particles which represent the current state. This approach is not suitable for low-end mobile robotics due to its requirement of high computation power.

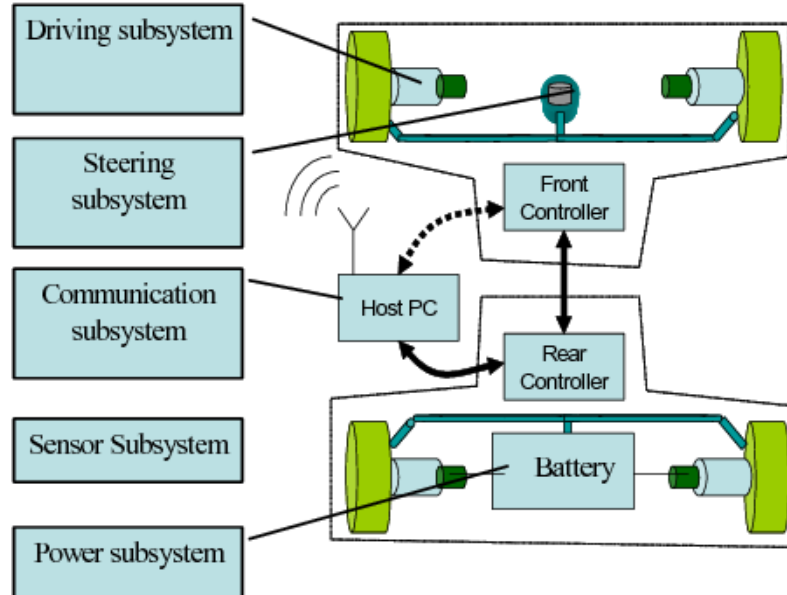


Figure 2.1: Subsystems [13]

Daniel Stonier et al [13] presents a model-based fault diagnostics and prognostics for wheeled mobile robots. Analytical models that represent nominal behaviors of the systems of the robot are created. Performance assessment is carried on system and

subsystem levels which facilitate multiple functionalities. for example, the power supply subsystem is a collection of all parts related to energy such as battery cells, wires, and circuits, etc., The residuals are obtained for every subsystem and the defective element is obtained. Sensors in a mobile robot play a crucial role in overall robot performance. Sensor data is important for robots that carry perception tasks for any malfunction or errors in sensor data may lead to failures. G. K. Furlas et al [5] presents a sensor fault diagnosis mechanism used for autonomous mobile robots. The internal sensor data is taken into consideration in this process. The Observer-Kalman filter identification technique is used for fault diagnosis. A group of observers is created and associated with each sensor of the mobile robot. The observers take the sensor data and control commands as input and publish residuals to detect faulty sensors.

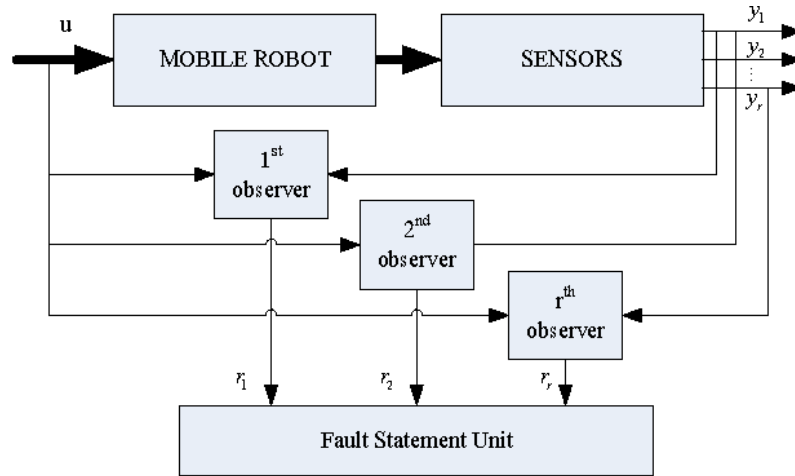


Figure 2.2: Observer residual generation [5]

Analyzing every subsystem of the mobile robot is a tedious task and developing a fault diagnosis system considering the overall behavior of the robot can reduce computation to a minimum. Pettersson et al [15] presents an execution monitor for behavior-based mobile robots. Fault detection and Isolation functionalities are hierarchically built inside a monitor. Fault isolation functionality is activated when a fault is detected (Figure 2.3). Activation levels of the robot are modeled and given to the monitor as input. Pattern recolonization technique is used to train the monitor by giving known fault states of the model. This method has a limitation of the number of fault models given for training.

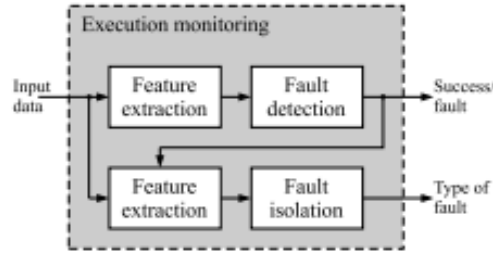


Figure 2.3: Execution monitor [15]

When it comes to reality there exists a lot of gap between the mathematical model and real hardware. The environment in which the robot moves change dynamically whereas, in a mathematical model, the environment that is considered is limited to few to zero variations. Mathematical model does not consider energy losses (mechanical losses) due to wear and tear in its kinematic control loop. Models are restricted to limited use case scenarios because the computational memory as well as the power increase as the variety of use cases are added. Furthermore, any addition of sensors in the future may lead to remodeling which costs time. In order to increase the accuracy of the detection of faults and considering these downsides of using a mathematical model, it is decided to work on an actual physical robot for this project.

A real robot is considered in which the main focus is given on monitoring the real-time behavior of the robot and predict for any faults. Faults can come from many reasons, the simplest of which if there is any obstacle along the path of the robot. Most of the automated robots are equipped with range sensors like Lidar, ultrasonic, etc., which can avoid any obstacles on the path. But what if there is a movable obstacle that is undetected by the range sensor? And a scenario where the robot grazes along the surface of a wall, falling into a hole and sliding along a slippery surface. Such scenarios cannot be detected using mere range sensors. For detecting these kinds of faults, sensor data of almost all the sensors present in the robot are to be evaluated. For example, in the case of slippery surface the actual distance transverse by the robot differs from the theoretical distance. So data from the velocity sensors help in identifying this behaviour. Taking the advantage of the fact that we are using a real robot, we plan on developing a run-time

monitor that receives that data given by the robot sensors and analyze the data to identify the fault.

2.2 Problem statement

In this project, the Kobuki robot, a widely established research platform based on an autonomous vacuum robot is used as a real robot for experiments and ROS - a state-of-the-art software platform is used as a developing environment to control the robot as well as extract the data from the robot sensors. Details of software architecture, hardware architecture, and implementation are explained in chapter 4. Two scenarios are considered and data from available sensors of the robot are collected. The data is plotted along time to get a limit value for differentiating between the cases.

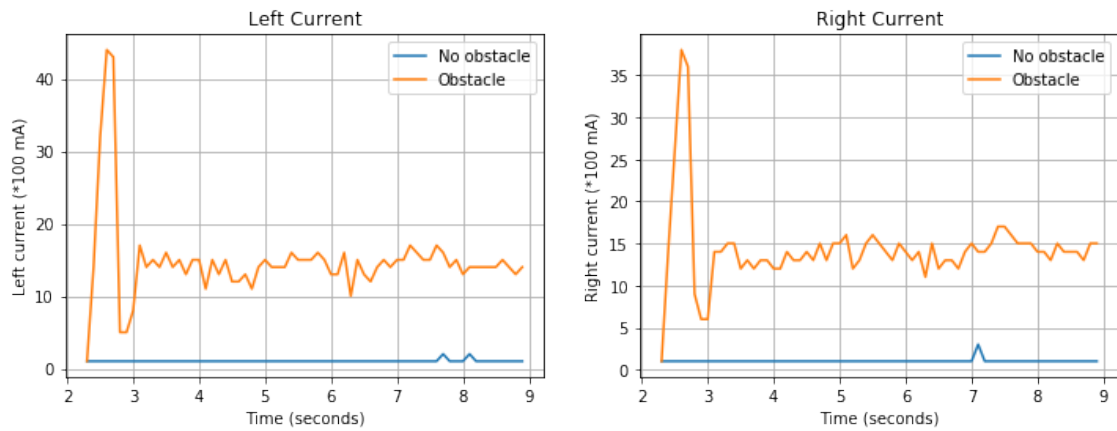


Figure 2.4: Binary class

To classify the data manually, a feature is selected and a limiting value for differentiating is obtained from analyzing the plots which is called Manual classification. Manual classification is efficient if the data can be separated easily and the nature of the problem is binary classification. If there are three or more classes, as shown in the figure 2.5 , it is difficult to manually obtain a threshold that can differentiate classes from one another.

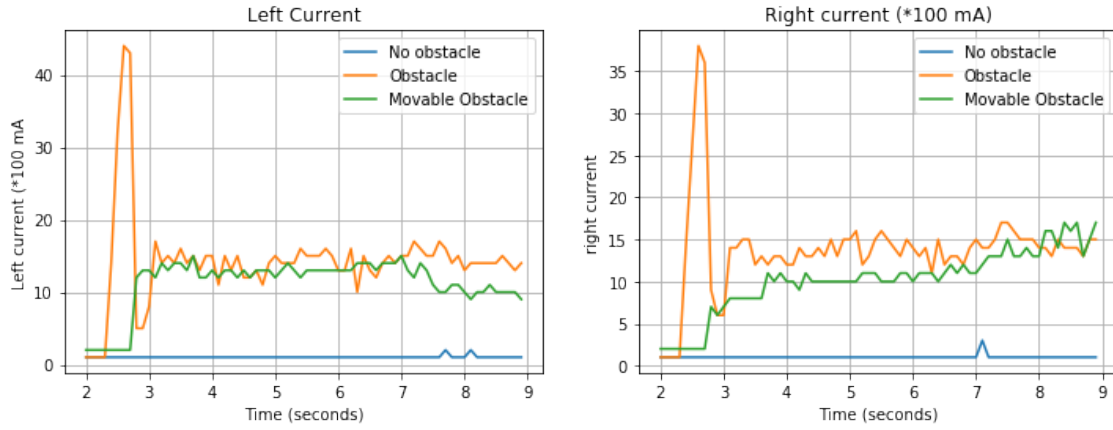


Figure 2.5: Ternary class

So, manual classification approach is not suitable for data containing three or more classes (chapter 5). Since the data from the robot consists of 6 different features and there exists a correlation between a few of this feature that may potentially increase the weight of classification accuracy. Manually developing an algorithm that considers all features simultaneously, find the correlation between the features, and classify the data is a hard task. Furthermore, manual labeling is only limited to the robot which is taken for experimenting. To use this kind of classification on other robots, one has to invest time and computation power on modifying algorithms to new parameters. All these downsides of manual labeling can be solved by using Machine Learning techniques. This problem is treated as a classification problem in classic Machine learning and we decided to solve this by implementing the Classification Model.

Chapter 3

Background

3.1 Machine Learning

”Humans learn from the past experience and machines follow instructions given by humans. What if humans can train machines to learn from the past data and do what humans can’t do at much faster rate” - Bernard Widrow 1959. Machine learning is a state-of-the-art development that revolutionized industrial processes and enhanced everyday living experience. Machine learning gives systems the ability to learn and improve from experience without being explicitly programmed [15] . It is a subset of Artificial Intelligence - A technique that enables machines to mimic human behavior. The concept of Machine learning was coined first by Alan Turning in 1950 in his article ”can machines think”. In 1957 Frank Rosenblatt designed the first machine learning algorithm to classify visual inputs, currently called the perception model. The Nearest Neighbour algorithm is written in 1967 which became the basis for pattern recognition algorithms in modern-day. In later decades, Machine Learning approaches are changed from knowledge-based to data-based. Currently, Machine learning is used in a variety of fields like medical, industrial, stock market, etc., in which major applications include Image and speech recognition, Medical diagnosis, classification etc.

Machine learning involves learning through analyzing the data, finding patterns, and making better decisions and predictions in the future based on data provided. Machine learning algorithms are broadly classified into three major types based on the type of data involved:

- **Supervised Learning** is a process where a machine learns under supervision/guidance.

It enables machines to predict outputs \mathbf{y} based on labeled data \mathbf{X} fed to the machine. Figure 3.1 depicts an example problem of supervised learning. Data of different objects and the names of the objects are given as input to train the machine learning model. The algorithm computes a mapping function \mathbf{f} that maps data and the labels. When the new data is queried, the algorithm uses that function to predict the label of the data. If the desired prediction is a continuous output variable \mathbf{y} for given input variables \mathbf{X} then the problem is considered to be the Regression problem. For example, a data of power readings of a motor and its operational velocity is given as training input to the model, and power consumption for a particular velocity is predicted, this is a regression task as power will be continuous output. If the desired prediction is a discrete output variable, then the problem is considered to be the Classification problem. If the data of features belonging to different cars and their corresponding brand names are given, the brand of a car is to be predicted when a new car feature is provided can be considered as a classification problem as the brand names are discrete in nature.

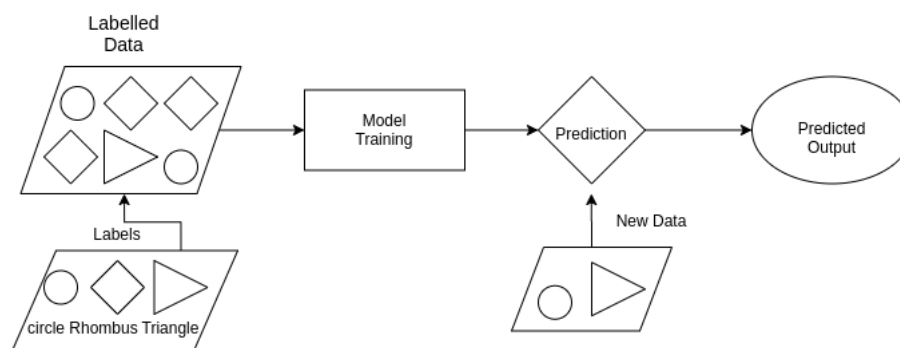


Figure 3.1: Supervised Learning

- **Unsupervised Learning** is a process where a machine is fed with unlabelled train-

ing data. The data is analyzed to find hidden patterns in order to make predictions about the output. Clustering is a type of unsupervised learning where patterns are found in the data which can be used to group the data. When data of different vehicles is provided and the algorithm analyzes the features of the vehicles and group them according to similarity basis, this type of grouping is considered to be a clustering problem. Association problem is another type of unsupervised learning method where dependencies between different data items are found. An anomaly detection problem analyzes patterns in the data and detects any anomalies or unusual activities in the data.

- **Reinforcement Learning** refers to goal-oriented algorithms that learns to reach goal using a series of actions. The initial state of the agent is given as input to the model. The input is based upon the actions taken by the agent to reach the final goal in an unknown environment which is called exploration. For each exploration step, rewards and punishments are assigned to the agent whether the goal can be reached using that step. The optimum solution is modeled on the basis of maximum reward steps. For example, when a robot is given a task to reach a goal of avoiding all obstacles, the robot starts with exploring the environment. Every action steps, its rewards are stored and a series of actions that can avoid all obstacles is the output of the algorithm.

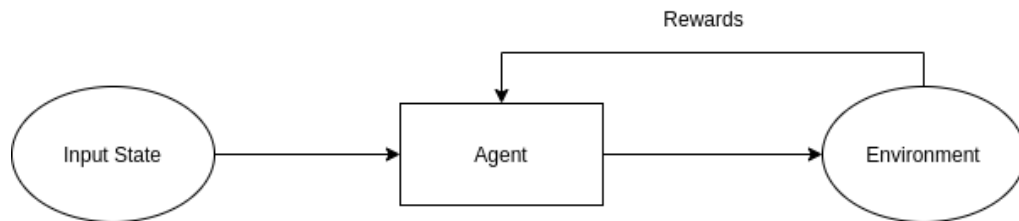


Figure 3.2: Reinforcement Learning

3.1.1 Nomenclature

A Machine learning algorithm is selected based on the objective of the problem and the type of data available. In this project, we are dealing with real-time sensor data

from a mobile robot. The main objective of the project is to diagnose faults which is a discrete output and the input has various sensors that constitute different features of the input data. So a classification problem is best suited for our problem statement. The terminology used in this section are :

- **Machine Learning Model** is a mathematical representation of the given input data which can map inputs to predictions (output) [12].
- **Machine Learning Algorithm** is a set of instructions or functions that are used to learn a machine learning model using data provided.
- **Data** refers to a table of data with rows and columns from which a machine learns. Rows correspond to individual data points or instances and columns correspond to different features of the information. Training data refers to the data used as an input to a Machine learning algorithm used to train a Model. Test data refers to the data which is used to test the accuracy of the model after successful training.
- **Classification threshold** is the boundary probability value to differentiate between two possible classification outcomes.
- **Labels** are the representation of classes. Training data is fed to the machine learning algorithm containing class data with corresponding class labels. The output prediction of a Machine learning algorithm is in the form of labels.
- **True Positives TP** are the number of actual positive instances that the machine learning model correctly predicted as positive.
- **True Negatives TN** are the number of actual negative instances that the machine learning model correctly predicted as negative.
- **False positives FP** are the number of actual negative instances that the machine learning model predicted as positive. These are also termed as Type I error.
- **False Negatives FN** are the number of actual positive instances that the machine learning model predicted as negative. These are also termed as Type II error.

- **Recall** is also known as sensitivity or True positive rate measures how "sensitive" the classifier is at detecting positive instances (add reference). It is mathematically represented as the ratio of True positive instances (TP) to all actual positive instances (TP + FN).

$$Recall = \frac{TP}{TP + FN}$$

- **Precision** measures how precise is the classifier in predicting positive instances ie., the accuracy of positive predictions. This is the ratio of True positive instances (TP) to all predicted positive instances (TP + FP).

$$Precision = \frac{TP}{TP + FP}$$

- **False Positive Rate** is also known as the false alarm ratio. This is the ratio of the number of falsely predicted negative instances to the total number of actual negative instances.

$$FPR = \frac{FP}{FP + TN}$$

- **f1-score** is a performance metric used for comparing two classifiers. It is a weighted average of precision and recall. This is a good metric when the data set is unbalanced ie., the number of instances for different labels is uneven.

$$Precision = \frac{2 * Precision * Recall}{Precision + Recall}$$

- **Accuracy** is another performance metric used widely to access the capability of a classifier. This is the ratio of the number of correctly predicted instances to the total number of instances.

$$Recall = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Average** of precision and recall values of individual classes in a multi-class classi-

fication is also a good metric for classification performance.

- **AUC - ROC** is an acronym for Area Under the Curve of Receiver Operating Characteristics curve. It is commonly used performance evaluation metrics for a binary classifier. This tells how capable is the model in distinguishing between two classes. ROC is a graph plotted between Recall (TPR) and False positive rates on the y-axis and x-axis respectively for various decision threshold settings. The Area under the curve (AUC) refers to the separability factor of the classification.

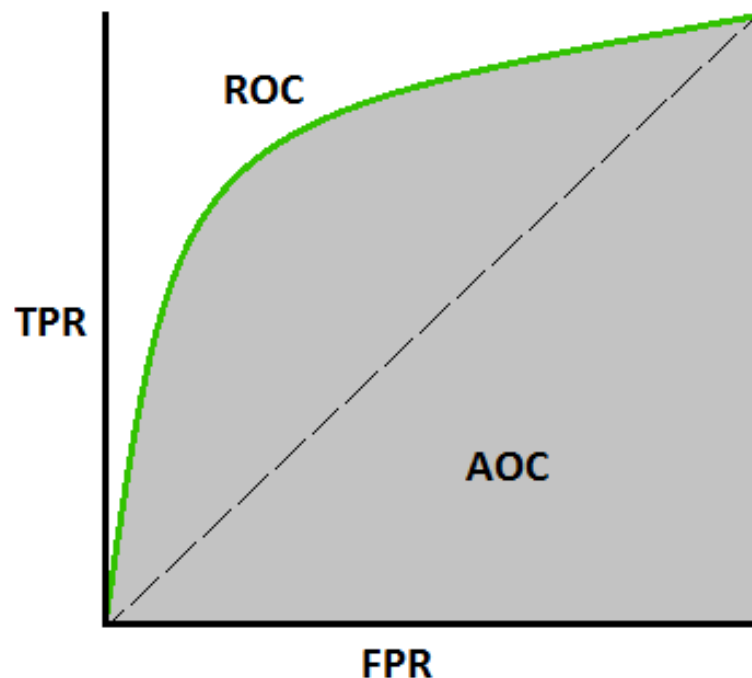


Figure 3.3: ROC Curve [14]

3.1.2 Classification problem

Classification is the problem of predicting the class of the new data point, on the basis of training data containing data points and their corresponding classes. When prediction involves a binary output, it is referred to as Binary classification. Binary Classification is used to predict one of two classes like yes/no, 0/1, etc.. When the data contains labels of two or more classes and a Label is to be predicted for a new data point,

a Multi-class classification model is used. The multi-class problem can also be solved using multiple binary class models. Multi-label classification is referred to as a type of classification problem where there exist multiple labels assigned to a single data point. Multi-class classification is referred to as a type of classification problem where a class is to be predicted from two or more classes. If a single data point is predicted to be assigned to one or more classes, a Multi-label classification is used.

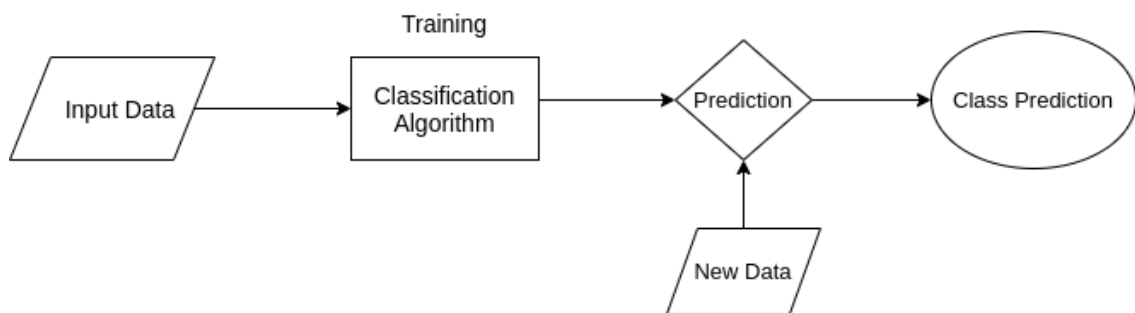


Figure 3.4: Classification workflow

There are different types of classification algorithms used in Machine learning. Logistic regression, Decision tree, Support vector machines, K-NN classification are majorly used algorithms for classification problems.

Logistic Regression is a kind of predictive analysis algorithm based on the concept of probabilities which is derived from linear regression. Linear regression is a method of fitting a line to a 2-Dimensional data with least squares approach. Linear regression is used to predict one quantity of data given the other quantity which is continuous. Logistic regression is similar to linear regression except the prediction is a discrete binary outcome (True/false). This is used for Binary classification containing two classes. Logistic regression evaluates the relationship between the dependent variable(label) and other variables (features) by fitting a logistic function. Logistic regression calculates the probability of a given data point belonging to the default class. Later these probabilities are passed through a sigmoid function to transform into binary values. Data points are classified to either 0 or 1 based on their probability relative to that of the decision probability (Default: 0.5). To use this classification for a multi-class problem, approaches like one to many or one to one are used which increases computation power and complexity

in analyzing the outputs.

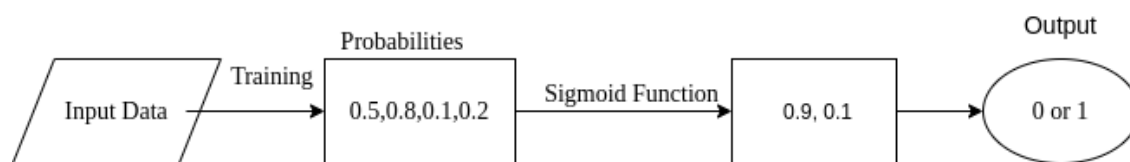


Figure 3.5: Linear Regression

Decision Tree Classification is a simple representation oriented classification algorithm used to predict the class variable by splitting the data in a tree structure. Each internal node or a decision node refers to a test that is implemented on an attribute of the data. Each branch represents an outcome of a test and the leaf or terminal node represents the class label. The topmost decision node is called the root node and it holds all the data. Data is split from the root node by creating branches for each possible outcome of the decision. This process is repeated recursively on every sub-node until all the data split from a branch belongs to a unique class label. This algorithm has a high sensitivity for the data. A small variation in the data can result in a completely different tree being created. Hence this cannot be used for data containing multiple features which may result in noise.

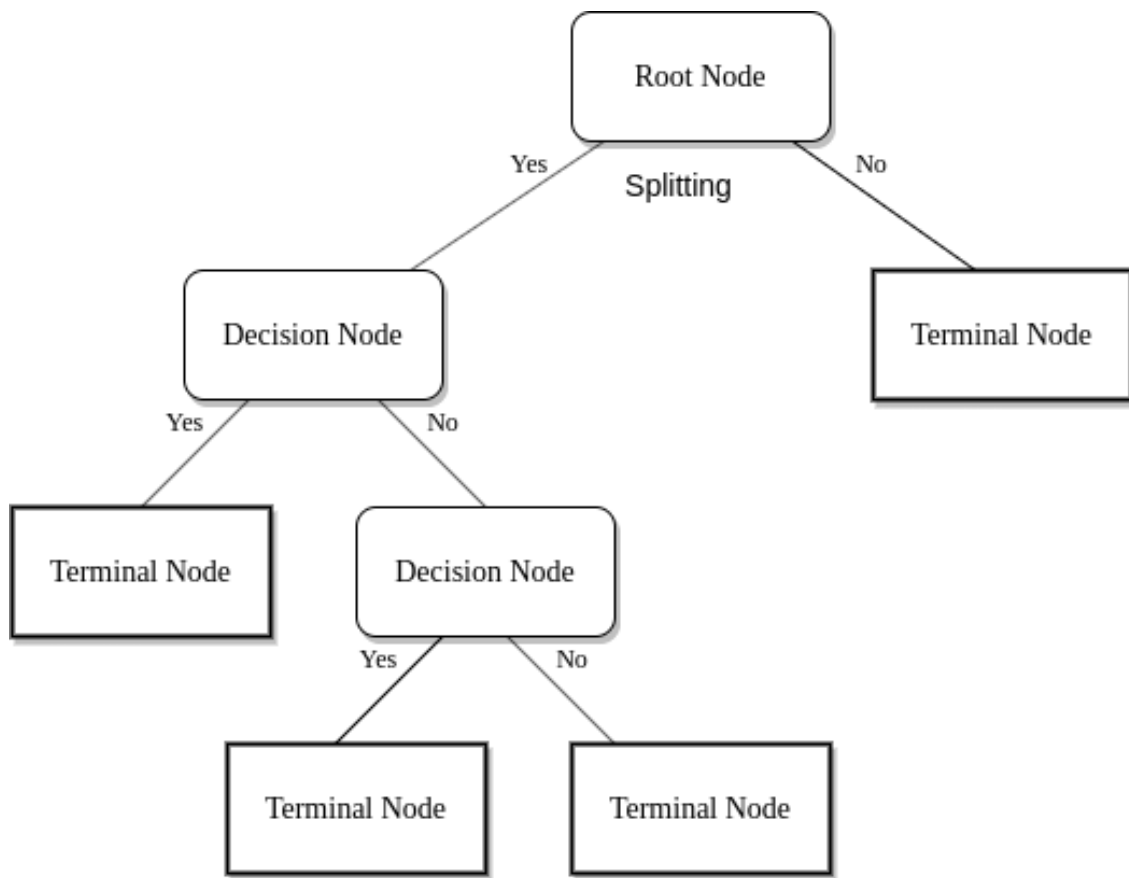


Figure 3.6: Decision Tree

k-Nearest Neighbors Classification (k-NN) algorithm classifies the data based on the similarity measure of features in the data. This algorithm classifies a new data point based on how its neighbors are classified. *Birds of a feather flock together* [8]. This is a memory-based algorithm. The training data with the features are stored in the memory when loaded to the algorithm. When a new data point is queried, this algorithm calculates the euclidean distance to all the trained data points available and sorts the distances in ascending order. 'k' in k-NN classification is a parameter that refers to the number of nearest neighbors to be considered. This algorithm takes first 'k' distances and gives the mode of the labels as the output. Figure 3.7 depicts an example representation of k-NN classification. k is an important parameter and the output changes based on the value of 'k'.

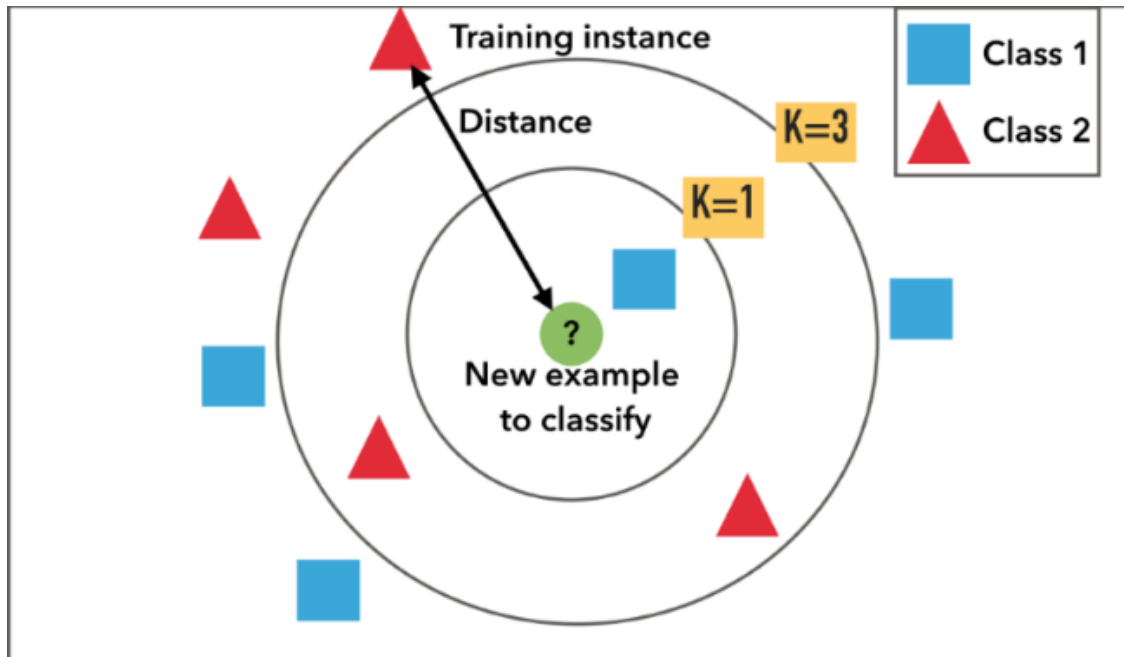


Figure 3.7: k-NN classification [1]

This algorithm is best suited for multi-class classification problems with multiple features because it is simple, intuitive, and easy to interpret. This algorithm is robust to noisy training data and effective if the data is large and has multiple features. The data from the robot comes from different sensors which are considered as features, they may be prone to noises. The main goal of this project is to develop a simple machine learning model for classification, this algorithm is selected for implementation.

3.2 Robot Operating System

Robot Operating System : Robots are used in real-time machines which are operated under dynamically changing environments where the user has to control the sensors and actuators accordingly. Software Development for robotics is quite challenging especially to make a code that can be used for different robots and sensors. As the scope of robotics is quite diverse as different types of robots have different hardware and they demand diverse software Implementations. On top of that, A robust code that

handles the entire life-cycle of robotics starting from driver configuration till high-end tasks like perception, logical reasoning needs to be developed. The user code needs to be easily maintainable as well as flexible for future Integration and Improvements. To meet these challenges, many robotic researchers collectively developed different robotic frameworks according to the requirements resulting in several Robotic development Environments (RDEs) [10]

ROS is one such RDE which is an open collaborative framework primarily designed to encounter challenges in the STAIR project [16] at Stanford University and the Personal Robots Program at Willow Garage [19]. Slowly researches started contributing their time and effort to creating core software packages. Many institutions began to develop the core packages for multiple robots. Now ROS evolved to be an open-source operating system which is a collection of tools, libraries and used for implementing complex and robust operations on a Robot. ROS handles communication in a peer-to-peer topology that simultaneously runs one to many processes on a number of different hosts. A single server can run different robots using different machines which are connected together via LAN or wifi.

ROS allows programmers to code according to their preferred language supporting C++, python, octave, and LISP currently. XML-RPC is used to implement peer-to-peer communication which is a stateless, HTTP-based protocol. It is a lightweight protocol that has its Implementations in different languages. Ros decentralizes the code into reusable standalone libraries. Most of the robotic drivers are programmed exclusively based on the middleware of the particular hardware which causes difficulty in the extraction of functionality and reuses it for different applications. Therefore all the development of algorithms is done in libraries that have no dependencies with ROS. Ros build system uses CMake to create executables that link library functionality to ROS.

3.2.1 ROS Nomenclature

The architecture of ROS is mainly composed of different components such as nodes, topics, services, etc.

Nodes are basic elements of ROS programming which perform actual functions for the robot. Each node in ROS has a name and is registered in ROS Master, i.e. it keeps track of every element running. Master is implemented via the XMLRPC server which handles all peer-to-peer connections between nodes and the Master. Nodes provide users a way to divide complex functionalities into simple programs. A combination of multiple nodes is called a nodelet which is often used in various robotic applications. For example, one node controls the robot arm motor while the other is responsible for image processing. A ROS node is programmed on top of ROS client libraries roscpp(c++) or rospy (python) provided by the ROS stack. Nodes communicate data through messages via topics. TCPROS protocol is used by nodes to establish a connection between nodes and topics. A publisher node publishes messages to the specific topic and these messages are available to the subscribers. A subscriber node gets the data from its subscribed topics.

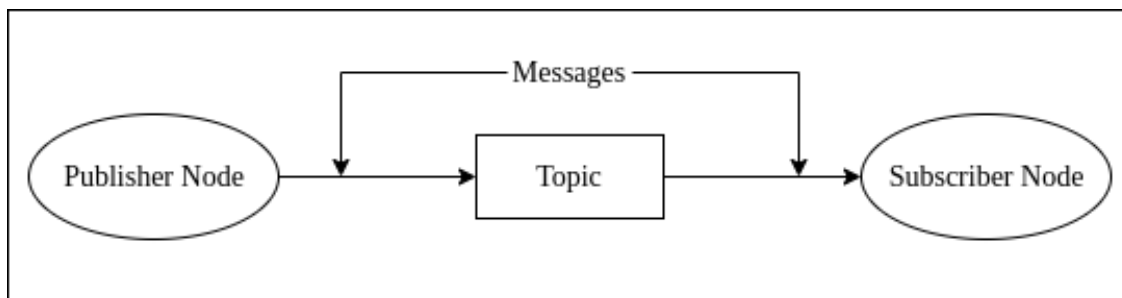


Figure 3.8: Node structure

Topics are uni-directional buses through which nodes exchange messages. Topics are implemented through anonymous sender/receiver semantics thus isolating the origin and destinations of the data being transmitted. Nodes that are interested in certain data are subscribed to corresponding topics and nodes that generate data are published to the relevant topic. It is possible to have multiple subscribers and publishers

to a node. Every topic transmits messages of the unique message type. Nodes that are subscribed/published to the topic should have a matching message type for successful communication. Topics use TCP/IP based communication protocol. TCPROS is a default transport that must be supported by client libraries of ROS.

Messages are the actual data structures in the communication layer of ROS. Nodes communicate through topics by messages. Messages are made of simple data types like integers, floating-point, boolean, etc. Ros provides default message types that are used commonly. Metadata of the message data structure are specified in simple text files called msg files which are stored in msg sub-directory of a package. Message files contain fields with the type and name (int32 x). A type defines the data type of that particular message and name is the display name of that message. Some messages use special fields called Header. Using header fix a frame id and timestamps to the messages. In the above figure 3.8, "cmdvel" is a topic that gets velocity data from "navigation" node in the form of "geometry_msgs/twist" message type. The structure of this message type is :

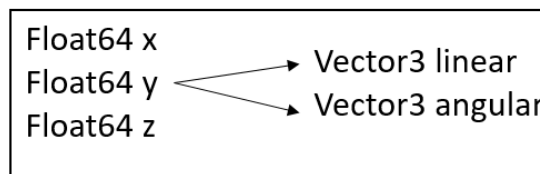


Figure 3.9: Fields

It has 6 fields of float datatype for velocities in 6 directions. "rosmmsg" commands are used to get information about the messages running in the ROS environment. ROS allows users to define custom message types.

As ROS was started as a development environment for the Willow Garage PR2 robot, it provided necessary software support for the research of PR2 robot and evolved to be useful for many applications. With the increase in the diversity of robots, the demands of the users are increased out of the range of ROS. The challenging use-cases that hinder usage of ROS are :

- **Multi-robot infrastructure:** Present architecture of ROS doesn't support multi-robot systems since ROS is a single-master environment. So, it is hard to integrate hardware devices into the ROS environment since the master is a central entity in the ROS graph and needs to be started before starting any node(corresponding hardware device) [17] .
- **Microcontrollers:** Microcontrollers are playing a crucial role in current state-of-the-art robotics for their compact yet efficient processing power. Communication between nodes is done through XML-RPC which needs heavy computation power to be used on microcontrollers. So, microcontrollers are isolated from ROS communication and a common driver is used to communicate between ROS and the microcontroller device. This may be a computationally good solution but communication speed is compromised. So, there is a need to include exposing ROS even on small resource systems like microcontrollers.
- **Real-time operation:** Operations are performed using node API and each node implements their own main function. Example: A node is defined for either publishing or subscribing, but the communication within a node is not possible. Some complex robots require real-time communication within certain processes. Also, the launching system of ROS1 can only initialize nodes simultaneously and there is no way to restart a node manually. In complex robotic applications, it is common to have time lags for different processes to start and there has to be flexibility to manage operations in runtime.

To overcome the above-mentioned backlogs and to increase the scope of ROS, ROS2 is developed as an upgrade to ROS1. The main advantage of ROS2 is its implementation of middleware for internal communication. All the communications in the ROS2 system are transmitted through ROS Middleware (rmw) which acts as a connection between ROS and DDS (Data Distribution Service). Ros middleware also acts as an abstraction to the DDS. ROS client library RCL was built on top of rmw which is a core ROS interface that implements logic and behavior of ROS concepts and is language

independent. Language-specific wrappers are built on top of RCL for the user to develop programs in c++ or python using rclcpp or rclpy respectively.

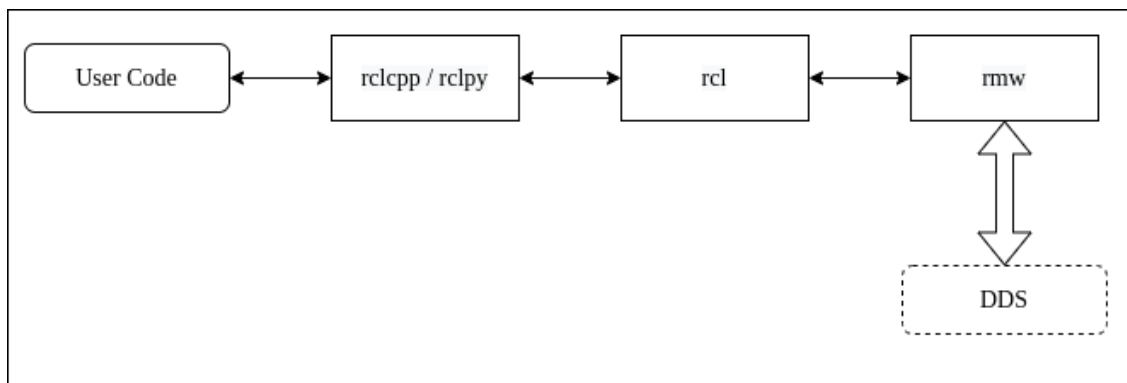


Figure 3.10: ROS2 Architecture

All the processes in the ROS system are stored and executed in the form of packages. A package can be interpreted as a container of user code. "package.xml" contains the metadata about the package which mainly includes details of the developer and dependencies of other packages in the ROS system. "CMakeLists.txt" file contains execution instructions of the user code. All the user code is stored in the src folder in the form of .cpp files. Different packages can be stored under a single workspace and can be built simultaneously.

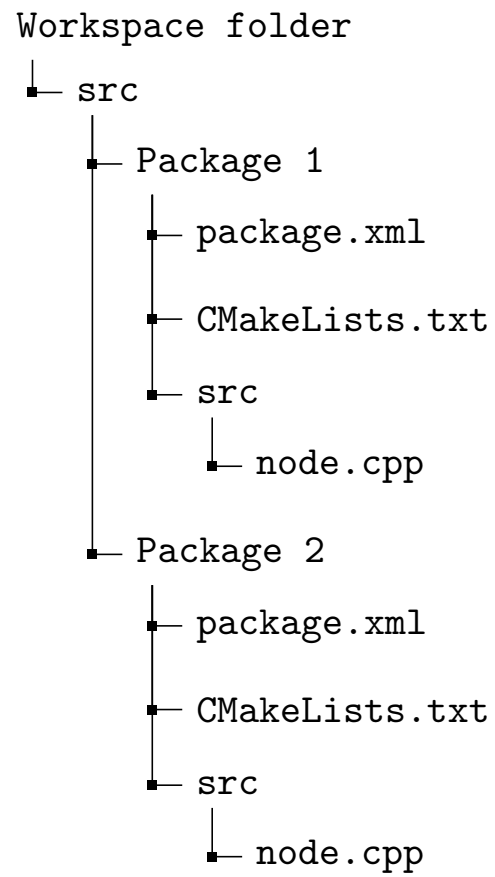


Figure 3.11: Folder Structure

Chapter 4

Design and Methodology

4.1 System Architecture

4.1.1 Kobuki Robot

Kobuki is a mobile robot designed for robotics research and development. It is a mobile base which has sensors embedded and motors for motion. The software driver communicates with the robot using a specified protocol. All commands and data are transmitted through a serial communication interface through byte streams [4]. Control commands for the robot are provided by the driver and the robot gives feedback about the required sensor data.

Maximum translational velocity	70 cm/s
Maximum rotational velocity	180 deg/s
Docking	Enabled
PC Connection	USB or via RX/TX pins
Odometry	2578.33 ticks/wheel rev
Power connectors	5V/1A, 12V/1.5A, 12V/5A
Sensor Data Rate	50Hz

Table 4.1: Technical Specifications [17]



Figure 4.1: Kobuki Robot

Kobuki has 2 Brushed DC motors with 12 V rated voltage and 750 mA rated load current. These motors are controlled by Pulse width modulation (PMW) based on the control command. Each motor is equipped with a wheel encoder which gives information about wheel odometry. Kobuki has a 3-axis Digital Gyroscope with a measurement range of ± 250 deg/s which provides the information about angular velocities and orientation of the robot. Kobuki has a bumper sensor in its left, right, and center areas surrounding the robot. Bumper sensors are spring-loaded and produce a specific value when one of the bumpers is hit (1: right bumper, 2: center bumper, and 4: left bumper). Kobuki also has a wheel drop sensor attached to both the wheels which send the feedback data in case of a wheel drop event. All the sensors feedback data is transmitted through byte streams. Header denotes the starting of a byte stream, length refers to the length of the bytes being transmitted in a single byte stream, the payload contains the actual data to be transmitted and a checksum is to check for any errors in the byte stream(Appendix A).

Headers		Length	Payload					Checksum
Header 0	Header 1		Sub-Payload 0	Sub-Payload 1	Sub-Payload 2	...	Sub-Payload N-1	

Figure 4.2: Byte Stream Structure [4]

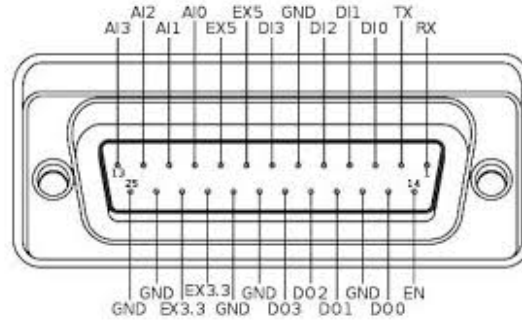


Figure 4.3: Serial pin-out [17]

4.1.2 Raspberry pi

Raspberry pi 3 is used as the low-level controller board which is attached to the robot and communicates with the robot. All the required ROS drivers and ROS packages are installed in the Raspberry pi. Quad-Core 1.2GHz Broadcom BCM2837 64bit CPU, 1GB RAM, BCM43438 wireless LAN, and Bluetooth Low Energy (BLE) on board are the important features of Raspberry pi 3B. The main goal of the project is to design a fault diagnosis system that is optimistic and that can be easily integrated with robot hardware. So, RasPi 3B is the best fit for our requirements. It communicates with the robot using USB serial communication (FTDI USB-Serial converter (FT232R)). It sends velocity commands to the robot through the ROS interface and makes the robot move according to the user's interest. It receives byte stream data from the robot through serial communication. RasPi 3B is connected to the kobuki robot's serial pins to power up. The serial pinout schematic is shown in figure 4.3. VCC and GND pins of the RasPi are connected to EX5 and GND pins of the Kobuki

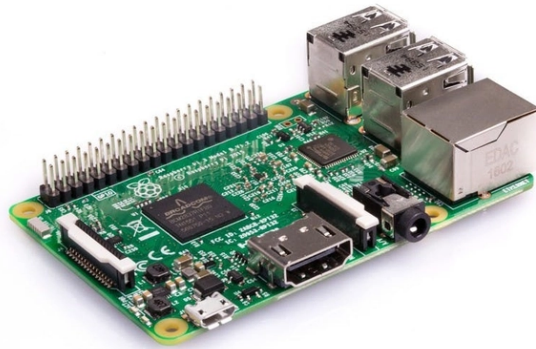


Figure 4.4: RasperryPi 3B

RasPi has inbuilt wifi hardware and performs wireless communication with any device that is in the same network. A Desktop is connected to RasPi using wireless communication serves as a monitor to RasPi. Users can perform tasks in RasPi using SSH protocol from the desktop.

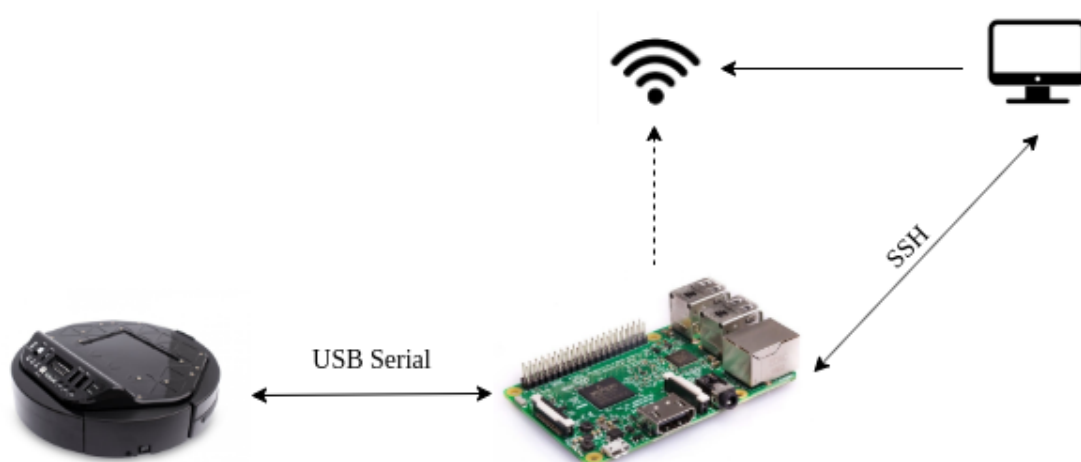


Figure 4.5: System Architecture



Figure 4.6: Hardware Architecture

4.2 Software Architecture

Ubuntu 18.04 Operating system is installed in Raspberry Pi and ROS Dashing version is installed which is used as a software interface for this project. All the required ROS packages for controlling Kobuki Robot are installed on RasPi. Core package that is used as the driver to control the robot is provided by Yujin Robots as an open-source ROS package called "turtlebot2_drivers". Kobuki_node is the main node that acts as a software communication interface between ROS and the Robot hardware. Kobuki node sends control commands to the robot hardware and gets feedback sensor data in the form of byte streams. It then converts raw data from robot to ROS message format through the Deserialization process. It publishes imu data through "/imu" topic as "sensor_msgs/Imu.msg" message type, odometry data through "/odom" topic as "nav_msgs/Odometry.msg" message type. A new message type "/current" is created which publishes data of the currents produced in right and left wheel, bumper, and wheel drop readings (Appendix A.2). The frequency at which data is been published can be modified inside the ros nodes and we decided to publish data at 20 hz frequency.

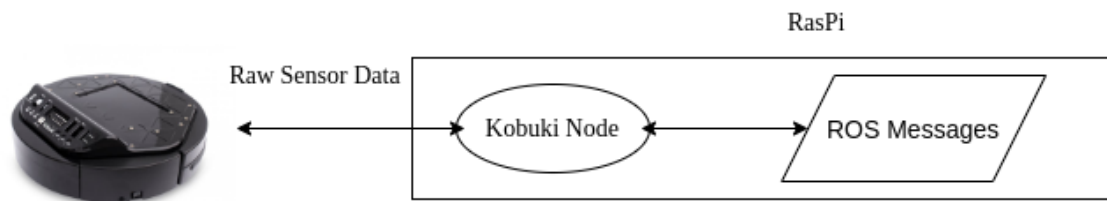


Figure 4.7: Kobuki node

A navigation node is created to send command velocities for making the robot move for a certain period of time (10 seconds). Navigation node is designed to publish command velocities through `"/cmd_vel"` topic as a `"geometry_msgs/Twist.msg"` message type. This node publishes velocities at a frequency of 20 hz. The kobuki node is subscribed to `"/cmd_vel"` topic and receives command velocity data. Kobuki node converts ros message data of command velocity into byte streams of raw data and send it to the robot which makes the robot move. As feedback, the robot sends the updated sensor readings and the kobuki node publishes those readings in the form of ROS topics.

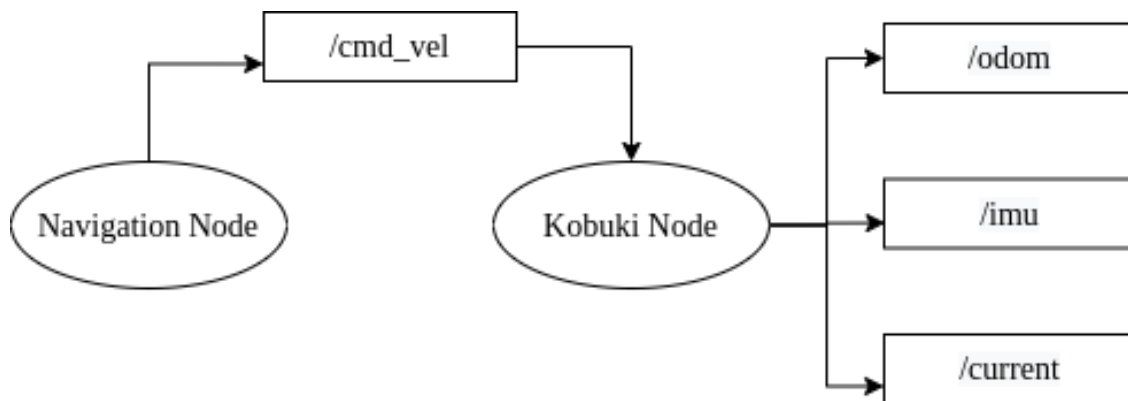


Figure 4.8: Navigation Node

The sensor data for each process must be saved in the disk for processing. A package names rosbag is used to record the entire flow of operations in bag files with .db format. It records all the topics and corresponding messages published on those topics with time. rosbag has python APIs in ROS1 to perform reading and writing operations on bag data. But that feature is removed in ROS2 Implementation. So, a data extractor node is created to extract the data from the ROS system and transform it to a user-readable

format for further processing. This node subscribes to the topics which are essential i.e., odom, imu, current, and velocity. When a node publishes some data to the topics, the extractor node receives that data from the topics and writes the data to a excel file using file streams provided by C++. The output excel contains ros message data written along the time. We use standard Unix epoch clock ¹ for saving time.

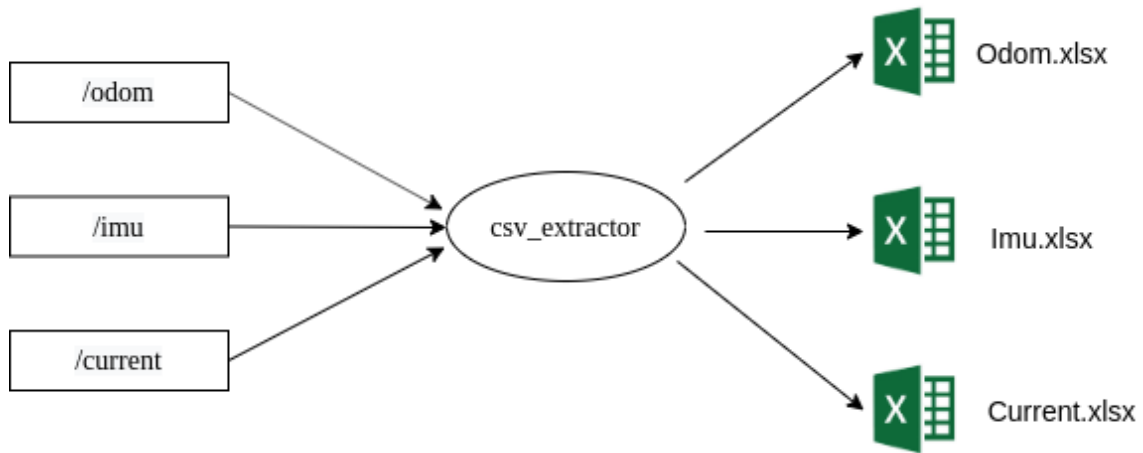


Figure 4.9: Csv Extractor Node

The user readable sensor data is then formatted and fed in to machine learning algorithm (section 4.3.2). A ROS python node "validation" is created where the trained model is loaded. Validation node is responsible for the run time prediction of faults during the motion of the robot. When the robot is moving, validation node subscribes to all sensor data from the robot and then process these sensor data as a input to the machine learning model which gives the prediction as output. This prediction is published to a ROS topic named "prediction". This node takes the sensor data and gives the prediction at the rate of 50 Hz frequency.

¹The number of seconds that have elapsed since 00:00:00 UTC, Thursday, 1 January 1970. It is the most common standard time used in robotics applications

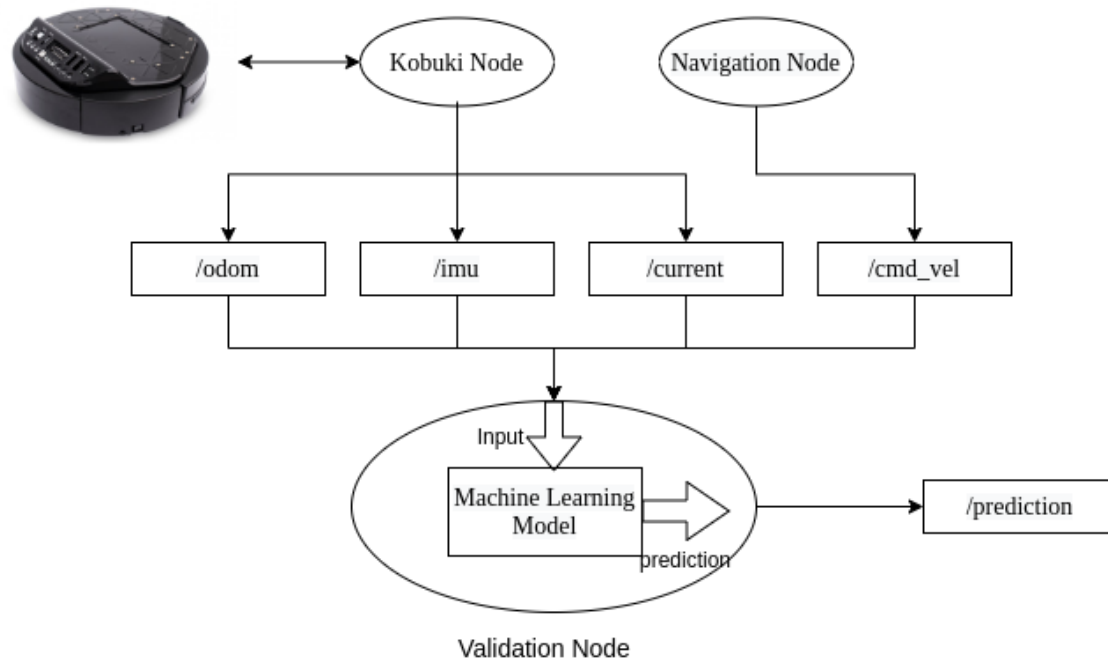


Figure 4.10: Validation node

4.3 Implementation

4.3.1 Use cases

Turtlebot2_drivers which contain software drivers for kobuki robot and robot navigation which is custom made package containing navigation node, CSV extractor node is placed in a workspace and build using colon build. A launch file is created to run both navigation node and kobuki node simultaneously (Appendix B). The robot is made to move for 10 seconds by publishing velocity to `"/cmd_vel"`. The kobuki node which is subscribed to the velocity topic makes the robot move and publishes the sensor data into respective topics. Ros2 bag is used to record the whole process. The recorder subscribes to the topics and records all the messages including the timestamps. This data is stored in the hard disk in the form of a `.db` file. The bag file is played in later stages, where the data extractor node is run as described in the above chapter. Thus the required data is

present in the form of user-readable excel files.

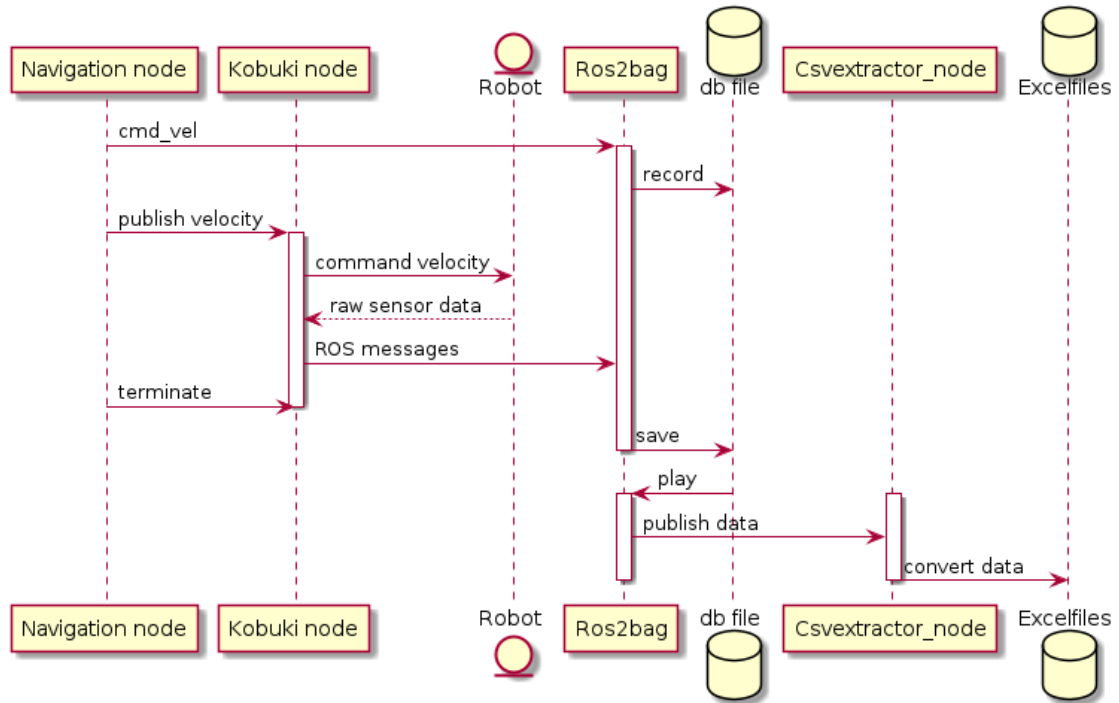


Figure 4.11: Ros Implementation

The main focus of our project is to identify faults that occur in a mobile robot which is been operated in domestic appliances. There are two steps involved in any machine learning algorithm to work, Training and Testing. One has to train the algorithm with the most probable use case scenarios. So, we came up with a few most common use case scenarios where a mobile robot encounters and the above mentioned ROS implementation is done for each scenario. The considered use cases are :

- (i) **No Obstacle** : The robot is made to move straight without any obstacles along the path. The data is extracted from the time of start till the time of stop. The
- (ii) **Rigid Obstacle** : The robot is made to move and along a rigid obstacle in the path. The data is extracted from the point where the robot touches the obstacle and recorded for 10 seconds of impact.

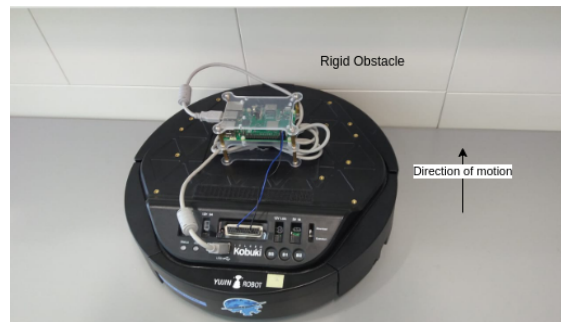


Figure 4.12: Rigid Obstacle

- (iii) **Movable Obstacle:** The robot is made to move straight and an obstacle that can be moved using robot torque is placed in the path of the robot. The data is extracted from the point where the robot touches the obstacles and recorded for 10 seconds.

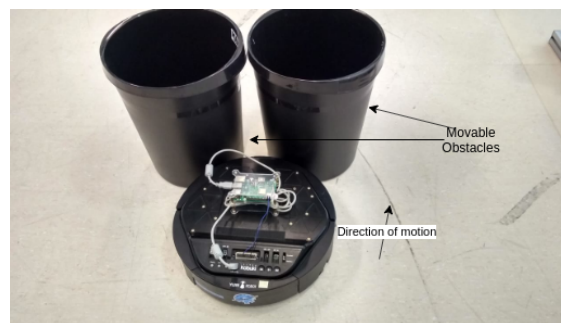


Figure 4.13: Movable Obstacle

- (iv) **Slip:** The robot is made to move and an obstacle is placed in the path, but a slip is created between the wheels and the surface. For this use case scenario, an artificial slip is induced to the wheels that reduce friction between wheels and the surface.

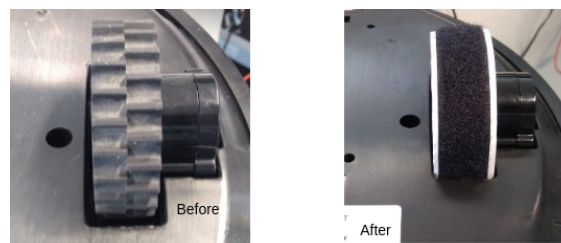


Figure 4.14: Slip

- (v) **Scratch:** The robot is made to move while scratching a plane in its motion. The

data is extracted from the point of time where robot touches the plane and recorded for 10 seconds of impact.

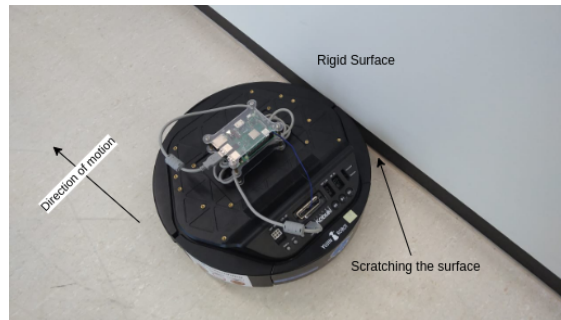


Figure 4.15: Scratch

- (vi) **Stuck:** The robot is made to move and a wheel (left/right) gets stuck in the path of the robot. Artificial resistance is created to the wheels using external material like tissue paper to replicate a scenario where a wheel gets stuck during the motion.

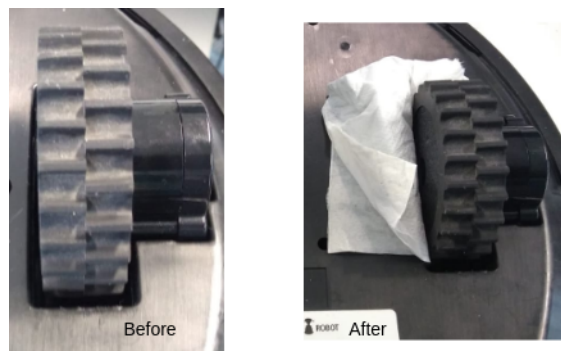


Figure 4.16: Stuck

- (vii) **Wedge:** The robot is made to move and it moves into a declining wedged surface and gets stuck when it is deep inside.



Figure 4.17: Wedge

- (viii) **Pit:** The robot is made to move and a pit in the path is created where wheels move freely because of the pit. The robot is made to move and a scenario where its wheel drop is created to replicate a pit.

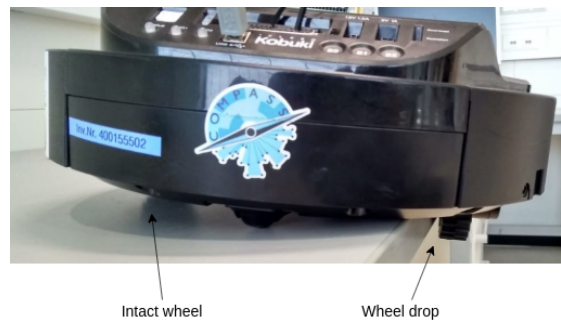


Figure 4.18: Wheel drop

After extraction, all the data from required sensors are merged together into a single Excel sheet which consists of columns Time, Command velocity, Actual velocity, Angular velocity, Right current, Left current, Bumper and Wheel drop. The data for each use case is visualized using python plots (Appendix B.2). Each row is the value of the respective parameter at the corresponding time step. After every use case, we manually label the data for preparing the data to train the model. The labels are 0-No obstacle, 1-Rigid Obstacle, 2-Movable obstacle, 3-Right wheel stuck, 4-Left wheel stuck, 5-Slip in the path, 6- Scratching along the surface, 7-Moving under the wedge, 8- Pit. After manually labeling the data, all the 8 excel sheets are merged into one which is used to train and test the algorithm.

Label	Use case Description
0	Obstacle
1	Rigid Obstacle
2	Movable Obstacle
3	Right Wheel stuck
4	Left Wheel stuck
5	Slip
6	Scratch
7	Wedge
8	Pit

Table 4.2: Description

4.3.2 Machine Learning Implementation

Machine learning implementation is done using the python programming language. scikit-learn is an open-source library that provides simple and effective tools for data analysis is used in this project. scikit-learn is maintained on top of core python libraries NumPy, SciPy and matplotlib. Jupyter notebook is used as a programming interface to implement scikit-learn libraries. The steps involved in implementing machine learning are as follows (Appendix B).

Step1: Obtaining and Analyzing the data

First, all the required libraries are imported to a jupyter notebook. NumPy is the base package for computations involving python. pandas give access to major data analysis functions in python and matplotlib is used for plotting purposes. The excel sheet containing labeled data is loaded as a pandas dataframe object. The errors in the data are cleaned before implementing the machine learning algorithm. For any use case scenario, the robot is started first and before it is impacted with obstacle , the robot is logically moving in no obstacle case. So, first few data points for every use case data set are removed.

Analyzing the data is an important step in training a machine learning model.

This step gives insights into the data. Macro parameters like mean, standard deviation, limits, etc., of the whole data, can be known using functions from the pandas package. Then correlations between the features of the dataset are to be determined. Pearson's Correlation Coefficient is used to find out the relation between two attributes. The values range from -1 to 1, 1 means high correlation and 0 means no or zero correlation. A 2-Dimensional heat map provided by *seaborn* package of python is used for determining and plotting correlation analysis.

Step2 : Formating the data

After analyzing, the data is divided into two partitions called Training and Test datasets. The data is split in the ratio of 3:1 where three quarters are used for training the machine learning model and one quarter is used to test the accuracy of the trained model. `X_train` contains all the data instances along with the features as columns and `y_train` contains corresponding labels of data instances. `X_train` and `y_train` are fed to the model for training as labeled data. `X_test` and `y_test` are used for testing the model.

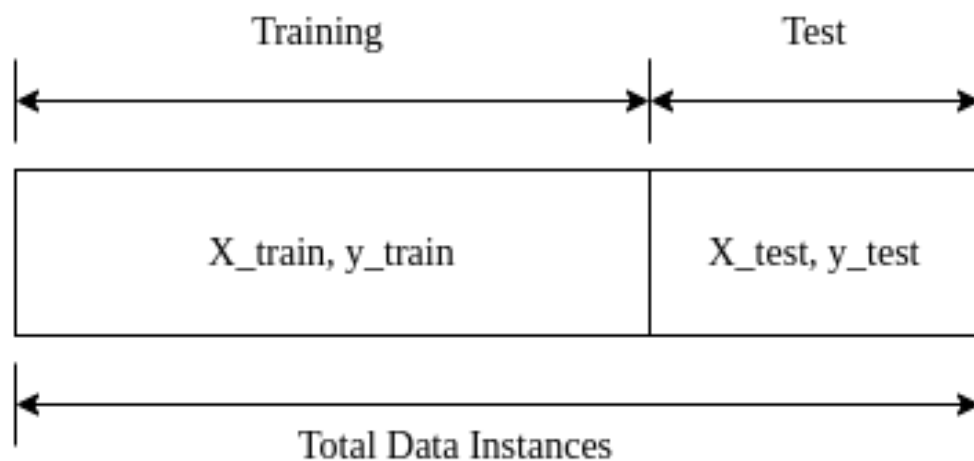


Figure 4.19: Splitting

Different features in the dataset have different magnitudes, units, and range. Current values from the motor are extracted in 10mA units whereas Bumper and wheel drop readings have no units. Knn classification uses euclidean distance (refer back-

ground) on the dataset to obtain a class prediction. So, features having high magnitudes tend to weigh more compared to features having low magnitudes and this can result in getting inaccurate predictions. To avoid this effect, all the features should have the same level of magnitudes and can be achieved by scaling the data. The data is scaled using a standard scalar function available in pre-processing libraries provided by sklearn. The data is scaled to zero mean and unit variance ($\sigma=1$). This uses below function to scale the data:

$$x' = \frac{x - \bar{x}}{\sigma} \quad (4.1)$$

where:

x is the initial data instance,

x' is the scaled data instance,

\bar{x} is the mean of the feature data and

σ is the standard deviation of the feature data.

Step3 : Modelling and Interpretation

Once the data is preprocessed, a knn-classification model is trained using a training dataset. This is performed by using the KNeighborsClassifier method provided by sklearn libraries. In training, the model is fitted with the training data which stores all the data points. The trained model is used to predict the output for all test data instances. The model computes euclidean distances for every data instance of the test dataset and gives the prediction as output and an accuracy score is obtained. The number of neighbors is an important parameter for the classifier. changing its number will affect the score of the classifier. Based on the scores that are plotted, an optimum number of neighbors is chosen. After choosing the number, the classifier is refitted with new parameters.

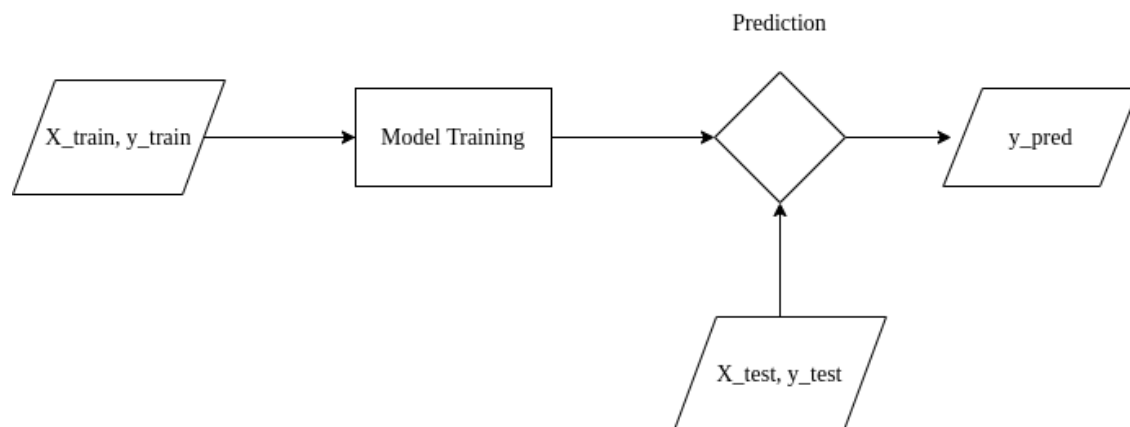


Figure 4.20: Training and Testing

X_{test} and y_{test} are used to test the accuracy of the model. The trained model is used to predict the output for data points that belong to X_{test} and the results are compared to y_{test} (the actual output). As mentioned in (section) for a binary classification problem, **True positives** are which the predicted output of the data points matches the actual label i.e, if the output is yes when it is truly a yes, **True negatives** are which the predicted output is no when the actual output is no, **False positives** are also known as type 1 errors where the classifier predicts yes when the data point is actually no i.e., the classifier predicts that there is an obstacle where there actually is no obstacle and **False negative** is also known as type 2 errors where the classifier predicts no when the data point is actually yes i.e., the classifier predicts that there is no obstacle when there is actually an obstacle. The confusion matrix is a tabular representation of all positives and negatives.

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Figure 4.21: Confusion Matrix

To interpret the accuracy of a multi-class classification, we define precision and recall for each class. For each class Precision means the accuracy of positive predictions i.e, it is the total number of correctly predicted data points out of all data points that are predicted to be this class, whereas Recall means a fraction of positives that are actually identified i.e, it is the total number of data points that are correctly predicted out of all actual data points belong to that class. The f1 score and accuracy are useful to know how often the classifier is correct.

The number of neighbors is an important parameter for the classifier. changing its number will affect the score of the classifier. Based on the scores that are plotted, an optimum number of neighbors is chosen. After choosing the number, the classifier is refitted with new parameters.

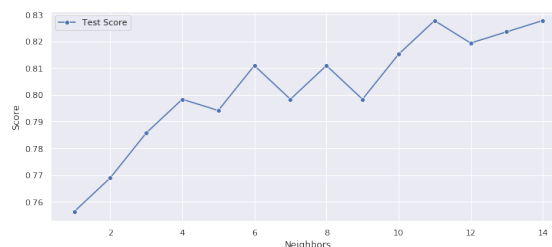


Figure 4.22: No. of neighbors

Once the classifier is trained and tested, the model is saved in the disk in the form of ".pkl" file. One of the motives behind using machine learning is it's re-usability.

Step 4 : Testing

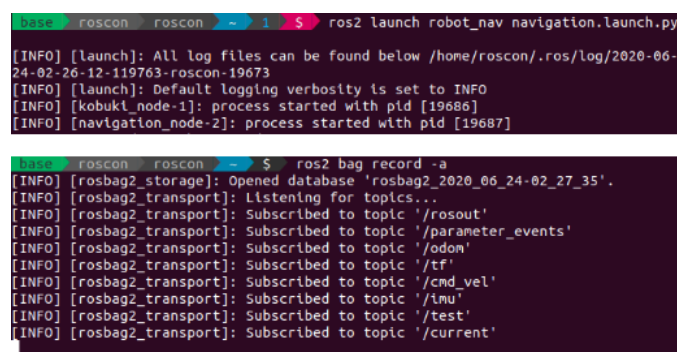
The saved model is loaded in to ROS validation node and tested in real time. The robot is made to move by launching kobuki node and navigation node. Validation node is also launched simultaneously. When the robot is moving, validation node which is running in the background collects the real time sensor data of the robot and publishes the prediction as output at a frequency of 1 Hz.

Chapter 5

Results

5.1 ROS Implementation

Kobuki robot is made to move by launching navigation.launch which initializes kobuki node and navigation node. A ROS bag recorder is launched simultaneously to record all the data flowing in the topics.



```
roscon roscon ~$ ros2 launch robot_nav navigation.launch.py
[INFO] [launch]: All log files can be found below /home/roscon/.ros/log/2020-06-24-02-26-12-119763-roscon-19673
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [kobuki_node-1]: process started with pid [19686]
[INFO] [navigation_node-2]: process started with pid [19687]

roscon roscon ~$ ros2 bag record -a
[INFO] [roslaunch]: Opened database 'roslaunch_2020_06_24-02_27_35'.
[INFO] [roslaunch]: Listening for topics...
[INFO] [roslaunch]: Subscribed to topic '/rosout'
[INFO] [roslaunch]: Subscribed to topic '/parameter_events'
[INFO] [roslaunch]: Subscribed to topic '/odom'
[INFO] [roslaunch]: Subscribed to topic '/tf'
[INFO] [roslaunch]: Subscribed to topic '/cmd_vel'
[INFO] [roslaunch]: Subscribed to topic '/imu'
[INFO] [roslaunch]: Subscribed to topic '/test'
[INFO] [roslaunch]: Subscribed to topic '/current'
```

Figure 5.1: Launching nodes

The recorded data is stored in the form of .db files. The recorded data is played and csv_extractor node is launched to extract the data in the form of excel sheets.

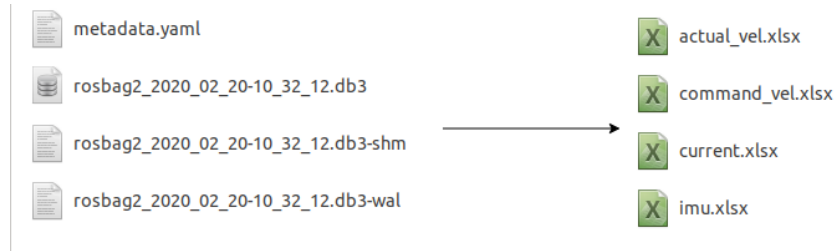


Figure 5.2: Data extraction

Data extracted from ROS is analyzed by plotting every feature against time. First a manual classification program that can classify a data point in to two classes is developed. The robot is run by launching ROS nodes in two use case scenarios. Data is extracted and labelled as 0: No obstacle and 1: Obstacle.

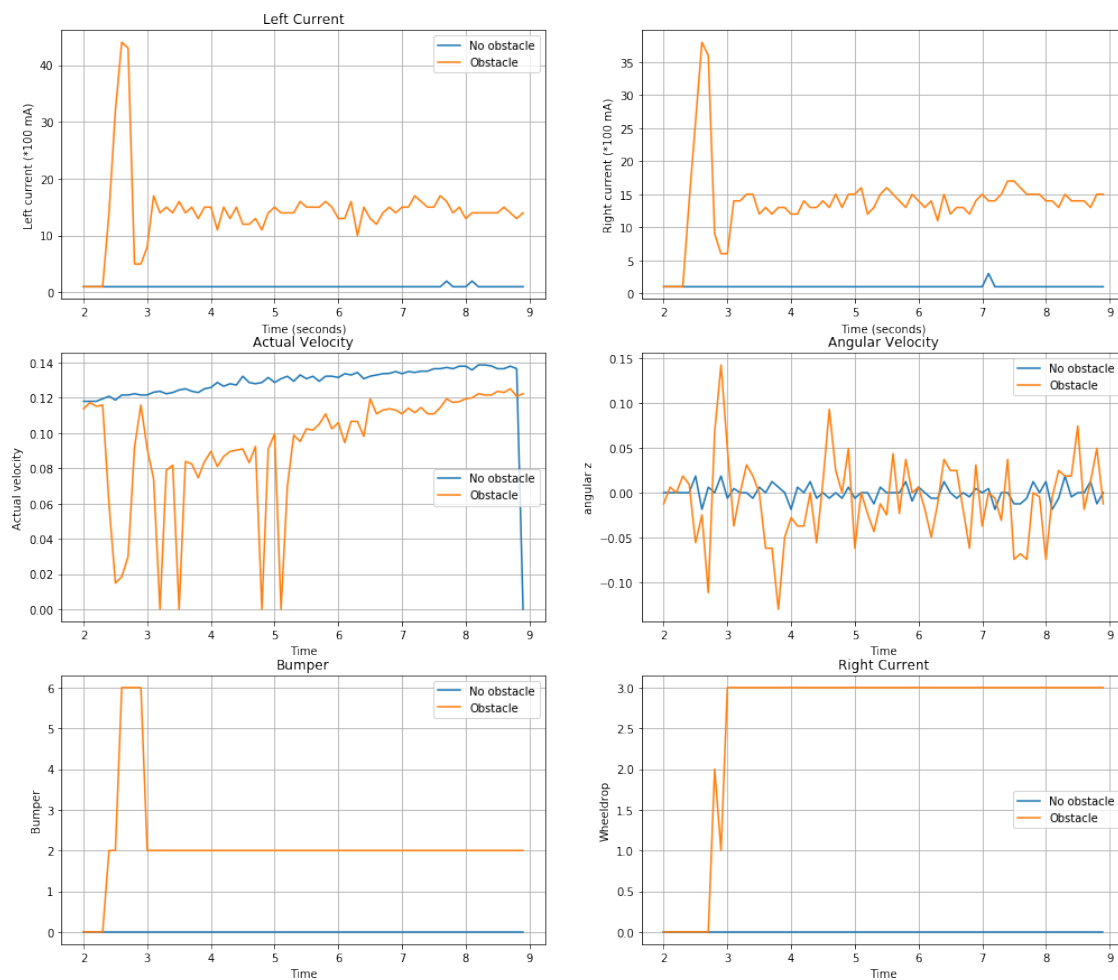


Figure 5.3: Feature plot

A discrimination threshold is decided to differentiate a data instance between two classes is derived by looking at the plot. Current readings are used as the discrimination feature. Data instances below threshold current are classified as '0' and above as '1'. Predicted labels are compared to actual labels to calculate the accuracy of the algorithm. True Positives, True negatives, false positives and false negatives (section 3.1.1) are calculated and a confusion matrix is plotted against those values and the obtained score of the algorithm is **0.67**.

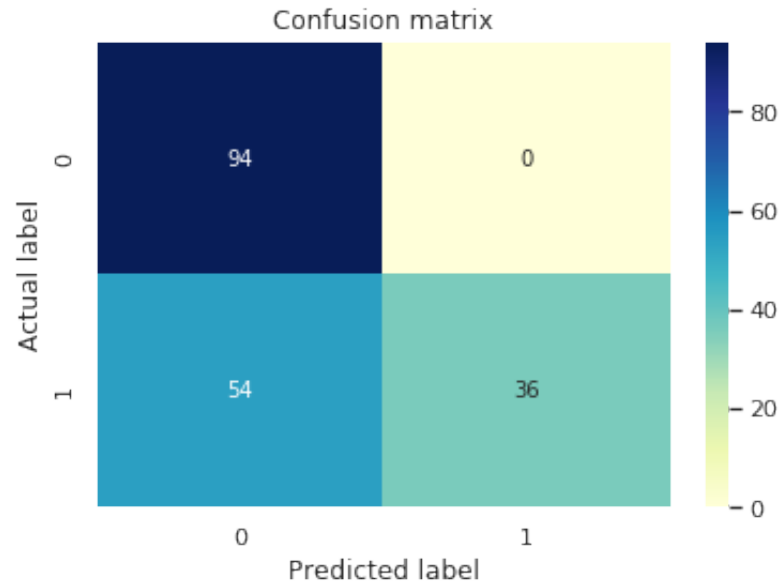


Figure 5.4: Confusion Matrix : Manual binary classification

A new class '2 : Movable Obstacle' is added for checking the accuracy of the manual classification algorithm. For this (refer section) robot is moved along a path with a obstacle that can transverse using robot's torque and the readings are extracted by following above process. Currents are plotted against time to obtain discrimination threshold for three classes.

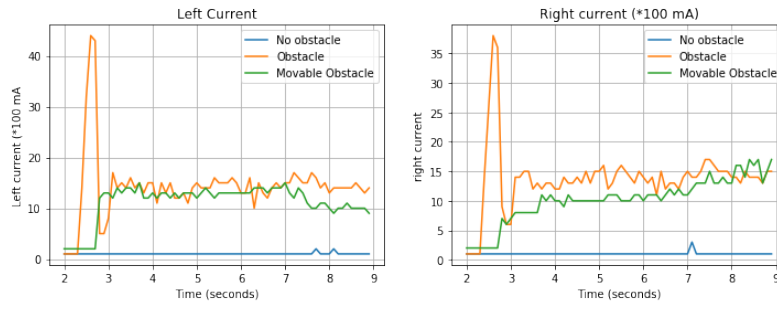


Figure 5.5: Three classes

The data is given to the manual classifier and predicted labels are compared to actual labels to get an accuracy score of **0.54** and the confusion matrix is shown in the figure. There is a fall in accuracy when added new classes to classify. A baseline classifier¹ is defined using the same input data and obtained a score of **0.52**. This result lead to use Machine learning algorithm for the implementation.

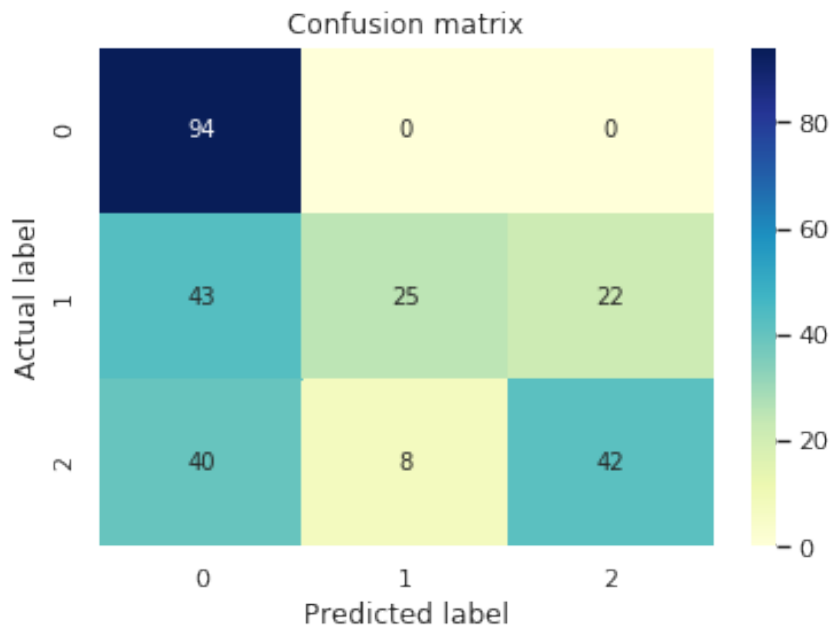


Figure 5.6: Confusion Matrix : Manual binary classification

¹A dummy classifier gives the minimum score that can be achieved by a Machine learning algorithm with simple rules.

5.1.1 Machine learning

Step 1 : Analyzing the data

The excel data that is extracted from ROS system is labeled manually with appropriate labels corresponding to the use cases. This excel data is loaded in to Jupyter notebook in the form of a data frame object. In the below figure dataset is the dataframe object created and the extracted data is stored in that object. Rows are the data instances that are recorded by incremental time steps and columns represent the features of the data

dataset								
	command_vel	actual_vel	angular_vel	Right current	Left current	Bumper	Wheeldrop	Label
0	0.15	0.000000	0.000000	0	0	0	0	0
1	0.15	0.000000	0.000000	0	0	0	0	0
2	0.15	0.056151	-0.006181	2	2	0	0	0
3	0.15	0.099507	0.000000	2	2	0	0	0
4	0.15	0.106615	0.012361	1	1	0	0	0
...
179	0.15	0.123674	0.074167	14	14	2	3	1
180	0.15	0.122963	-0.018542	14	15	2	3	1
181	0.15	0.125095	0.012361	13	14	2	3	1
182	0.15	0.120830	0.049445	15	13	2	3	1
183	0.15	0.122252	-0.012361	15	14	2	3	1

184 rows × 8 columns

Figure 5.7: Dataframe object

Once the data is loaded successfully, the data is cleaned by removing few initial data points.

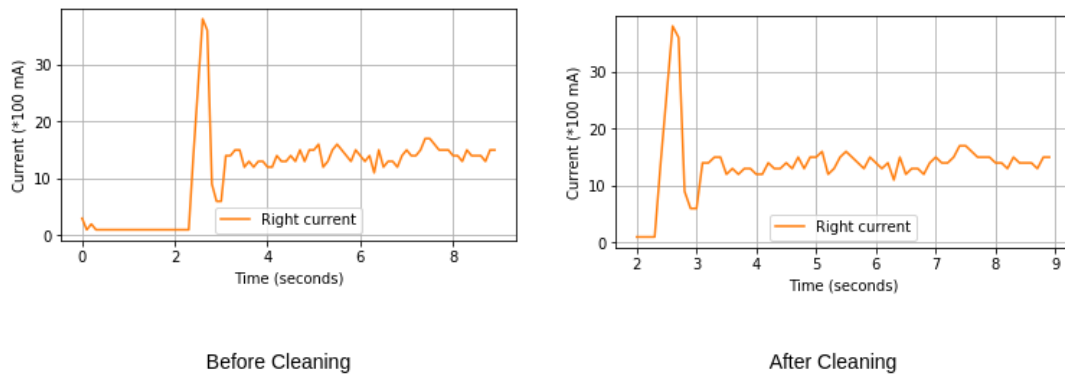


Figure 5.8: Cleaning

Step 2 : Formatting the data

After cleaning, the data is cleaned by removing any zero or null assigned data instances occurred during interpolation. Then the correlation is between the features is analyzed. The below figure shows there is a good correlation in between current and command velocity, both the currents, current and bumper reading features. This strengthens the hypothesis to use machine learning for implementing this project [section 2]

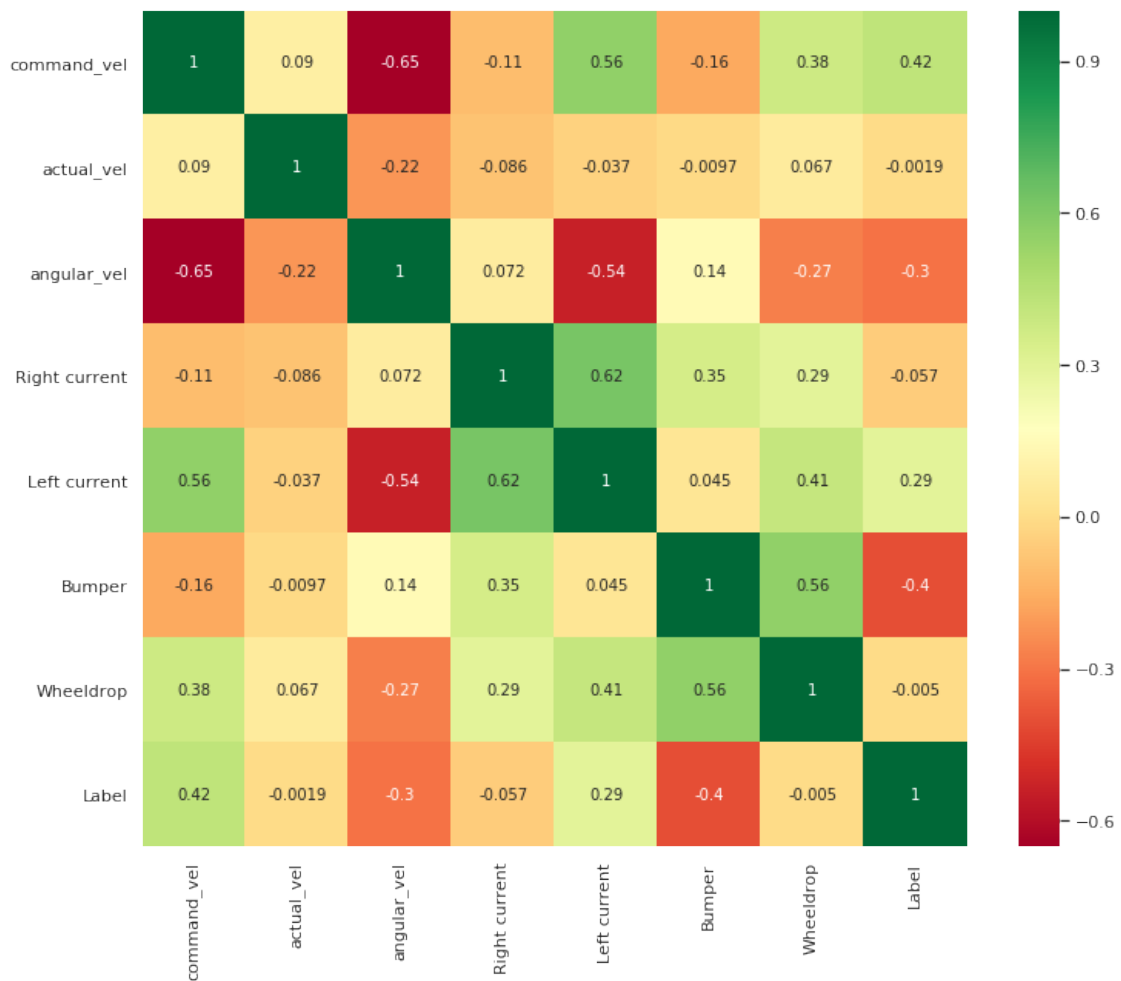


Figure 5.9: Correlation

Formatted data is split in to training and test datasets which are used for training and testing the model respectively. The dimensions of train and test datasets are displayed in the below figure

```
X_train.shape - (172, 7)
y_train.shape - (172,)
X_test.shape - (75, 7)
y_test.shape - (75,)
```

Figure 5.10: Data Split

Scaling is done on the data to fit data of every features in to zero mean and

unit variance. The magnitudes of data instances for corresponding features are changed (figure 5.9).

	command_vel	actual_vel	angular_vel	Right current	Left current	Bumper	Wheel drop	Label
0	0.15	0.000000	0.000000	1	1	0	0	0
1	0.15	0.117277	-0.006181	1	1	0	0	0
2	0.15	0.117987	-0.012361	1	2	0	0	0
3	0.15	0.000000	0.000000	1	1	0	0	0
4	0.15	0.117277	-0.018542	1	2	0	0	0

(a) Unscaled

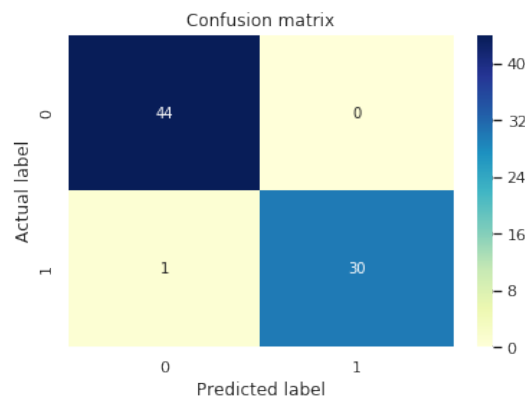
	command_vel	actual_vel	angular_vel	Right current	Left current	Bumper	wheel drop
0	-0.375509	-1.713372	0.270378	-0.497101	-0.595908	-0.462954	-0.618639
1	-0.375509	0.409893	0.248082	-0.497101	-0.595908	-0.462954	-0.618639
2	-0.375509	0.422747	0.225785	-0.497101	-0.533276	-0.462954	-0.618639
3	-0.375509	-1.713372	0.270378	-0.497101	-0.595908	-0.462954	-0.618639
4	-0.375509	0.409893	0.203489	-0.497101	-0.533276	-0.462954	-0.618639

(b) Scaled

Figure 5.11: Scaling

Step 3 : Modelling and Interpretation

A knn classifier model is defined and training data is fit to the model. The trained model is used to predict outputs for the test dataset. All the predicted outcomes are stored in y_predicted array which is compared with y_test to get accuracy report for the classifier. Knn Binary classifier produced an accuracy score of **0.97** which is a good improvement compared to the score obtained in manual classification (0.67).



(a) Confusion Matrix - Binary classifier

	precision	recall	f1-score	support
0	0.98	0.98	0.98	44
1	0.97	0.97	0.97	31
accuracy			0.97	75
macro avg	0.97	0.97	0.97	75
weighted avg	0.97	0.97	0.97	75

(b) Classification Report - Binary classifier

Figure 5.12: Output - Binary classification

Now this model is used for already extracted data of three classes by following above steps and the results are shown in the figure. The accuracy score in classifying three classes is obtained as **0.92** which is improved from that of manual classification (0.52).

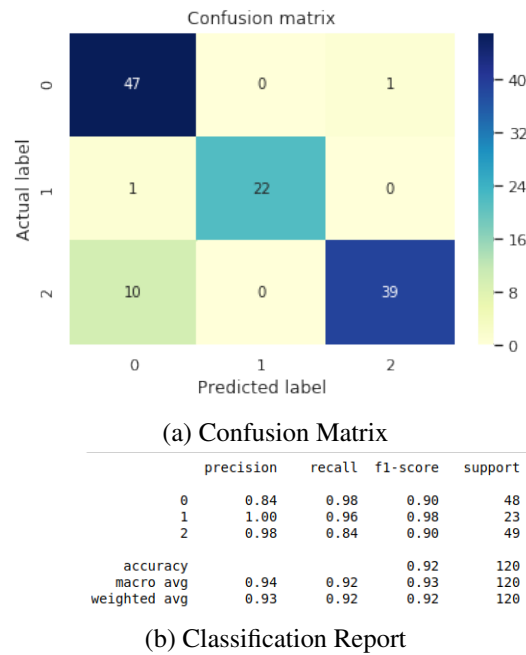


Figure 5.13: Output - Three classes

Robot is run to extract data from all the use case scenarios mentioned in section 4.3.1. Data is labeled with use case numbers (table 4.2) and merged together for implementation. Above steps are followed for machine learning implementation to achieve multi class classification. The confusion matrix and classification report for this 8 class classification is displayed below.

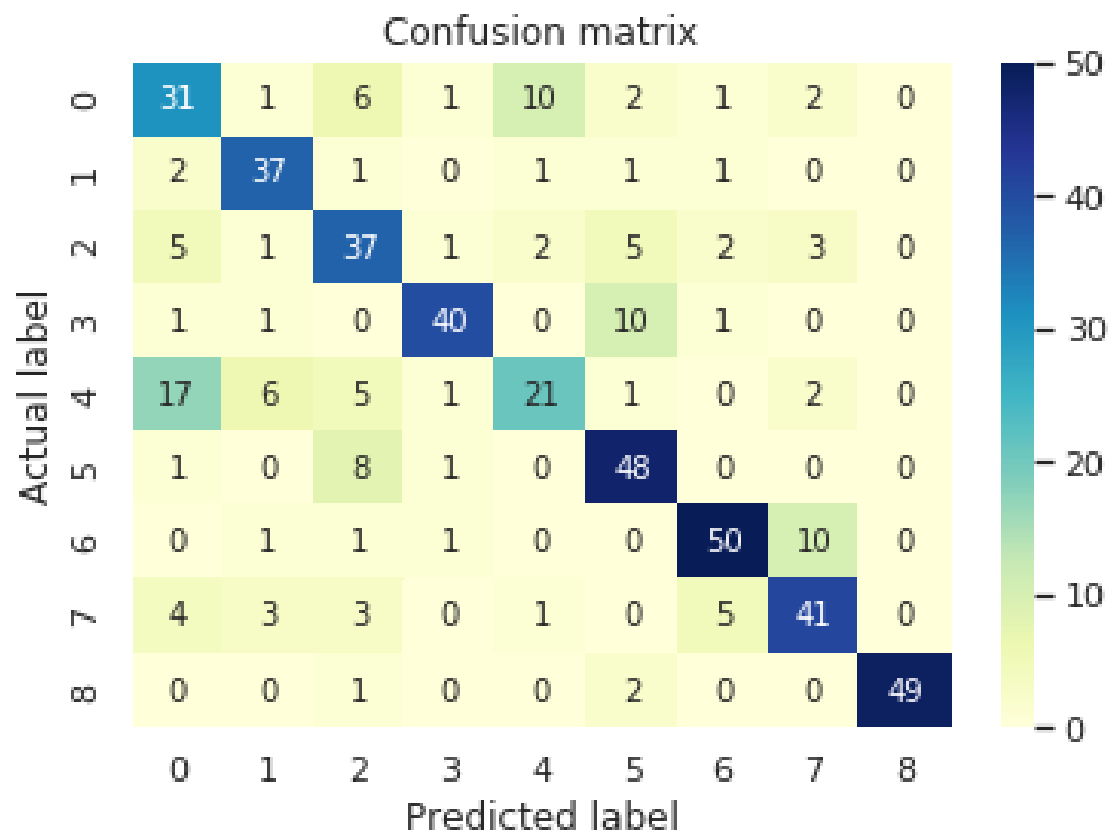


Figure 5.14: Confusion Matrix : Multi-class

	precision	recall	f1-score	support
0	0.48	0.74	0.58	54
1	0.77	0.84	0.80	43
2	0.76	0.61	0.67	56
3	0.79	0.91	0.84	53
4	0.72	0.49	0.58	53
5	0.75	0.84	0.80	58
6	0.89	0.78	0.83	63
7	0.85	0.72	0.78	57
8	1.00	0.94	0.97	52
accuracy			0.76	489
macro avg	0.78	0.76	0.76	489
weighted avg	0.78	0.76	0.76	489

Figure 5.15: Classification Report : Multi-class

Step 4 : Testing

The trained machine learning model is saved in to validation node. The robot is made to move and the prediction topic is captured in a time frame of 6 seconds. The validation node is tested for every use case scenario (Table 4.2) and the prediction results obtained for every second are captured. The prediction results are tabulated in the below figure 5.14.

Time	Label								
	0	1	2	3	4	5	6	7	8
1 sec	6	1	2	6	4	2	2	1	8
2 sec	0	1	2	3	4	5	6	1	8
3 sec	0	1	2	3	4	5	6	2	8
4 sec	0	1	2	3	4	5	3	7	8
5 sec	0	1	1	6	0	5	6	4	8
6 sec	4	1	2	3	0	2	6	7	8

Figure 5.16: Run time predictions

Chapter 6

Discussion

The project is started with a aim to contribute for the development in the field of mobile robotics. To design and implement a fault diagnosis system, selecting the software and hardware is crucial. The Kobuki robot is selected as it is the most commonly used robot for research and development. ROS is selected as the software platform for its simple yet effective performance in robot programming. The goal is set to use a basic machine learning algorithm on hardware with fundamental sensors like odom, imu and current, and test for its performance in real-time. After linking all the hardware together, one of the main challenges is to design a robust software architecture to manipulate the robot, fetch the sensor data and store the data in a defined time frame while the robot is in motion (manual intervention is not possible). Every task is independently programmed as nodes - navigation node to move the robot, extractor node to fetch the data. These nodes are run simultaneously to execute a specific use case scenario. Use cases that are most common in mobile robots prone to errors are selected and data is stored for these use cases. Obtaining the training data from raw sensor output data is a critical issue and needs more attention. For this, the extracted sensor data is cleaned and formatted. Python modules like pandas and matplotlib are used to analyze and format the data and scikit modules are used for training machine learning. Once the model is trained, the trained model is loaded into the software of the system and it is tested for its accuracy on

the Kobuki robot.

6.0.1 Conclusion

A fault diagnosis framework using kobuki robot which can be used in diagnosing faults of a mobile robot by analyzing the sensor data has been developed. The software and the hardware architecture of the system are implemented. The sensor data of the robot is extracted in 8 different fault cases and fed to a machine learning model for training. Then the trained model is embedded in software design and tested on a real system to predict the faults while the robot is in motion. The model showed good performance in detecting an obstacle (Figure 5.12, Figure 5.13) in its motion. Figure 5.14 depicts the performance of the model in detecting different types of use case scenarios. The used machine learning model (knn classification) classifies the data based on feature similarity. The readings from each sensor of the robot refer to a feature of the data. Features play an important role in the accuracy of the predictions.

Features of the data play an important role in the accuracy of the predictions. The model uses the features to distinguish a data point among different classes. A combination of different feature (sensor) data is used to distinguish a data point from one class to another. Correlation between features and class labels is a metric to tell how important is the feature in distinguishing a particular class. For example, to detect if the robot moved in a pit, the model can differentiate based on the wheel drop feature value of the data point. Similarly, the actual velocity feature could have been a distinguishable feature for most of the classes. Classes like a robot having a slip, and robot moving under a wedge contains current values in a close range and the actual velocities have weight in differentiating. The velocities are derived from wheel odometry which uses wheel encoders as sensors. Due to the errors in wheel odometry, the actual velocity is incorrectly obtained which leads to a decrease in correlation between velocity feature and the label of the class. This results in having a few similar data points for different classes which may be the reason for the model giving false predictions for some of the classes (Figure

5.14).

Most of the fault detection approaches till date are model based that consider mathematical models of subsystem to detect residuals which is complex and require great computation power. Important contributions of this work are :

- A behaviour based fault detection is designed and developed which focuses on the run-time behaviour of the mobile robot in a environment.
- The designed approach is able to detect variety of faults using basic machine learning algorithms with less computation power compared to fault detection approaches that use mathematical models.

6.0.2 Future work

The accuracy of the model depends on the sensor data. Most of the mobile robots use wheel odometry to get the position and velocities of the robot. To increase the accuracy of the odometry, a 9-axis imu sensor can be used to obtain velocities accurately. Some additional use case scenarios can be classified using this approach by adding some additional sensors. A mobile robot is most likely to move near a edge and preventing the robot to fall from a edge. A mobile robot can be prevented in moving along a undesirable land conditions like flow of water that may cause damage to internal circuits. For these kind of use cases, perception sensors like Lidar and Kinect camera can be incorporated.

A fault recovery system can be designed that take actions based on the type of fault diagnosed. Fault recovery system can be extended to the fault diagnosis system to make mobile robots fault tolerant.

References

- [1] Adi Bronshtein. *A Quick Introduction to KNN Algorithm*. <https://blog.usejournal.com/a-quick-introduction-to-k-nearest-neighbors-algorithm>. Apr. 2017.
- [2] J. Carlson and R. R. Murphy. “Reliability analysis of mobile robots”. In: *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*. Vol. 1. 2003, 274–281 vol.1.
- [3] D. Crestani, K. Godary-Dejean, and L. Lapierre. “Enhancing fault tolerance of autonomous mobile robots”. In: *Robotics and Autonomous Systems* 68 (2015), pp. 140–155. ISSN: 0921-8890. DOI: <https://doi.org/10.1016/j.robot.2014.12.015>. URL: <http://www.sciencedirect.com/science/article/pii/S0921889014003157>.
- [4] Jorge Santos Simon Daniel Stonier Younghun Ju. *Kobuki Driver Documentation*. <http://yujinrobot.github.io/kobuki/enAppendixProtocolSpecification.html>. Mar. 2017.
- [5] G. K. Fourlas, G. C. Karras, and K. J. Kyriakopoulos. “Sensors fault diagnosis in autonomous mobile robots using observer — Based technique”. In: *2015 International Conference on Control, Automation and Robotics*. 2015, pp. 49–54.
- [6] G. K. Fourlas et al. “Model based actuator fault diagnosis for a mobile robot”. In: *2014 IEEE International Conference on Industrial Technology (ICIT)*. 2014, pp. 79–84.

- [7] Dr Khasha Ghaffarzadeh and Dr Na Jiao. "Mobile Robots, Autonomous Vehicles, and Drones in Logistics, Warehousing, and Delivery 2020-2040". In: *IDTechEx* (2018). ISSN: 0921-8890.
- [8] Onel Harrison. *Machine learning- KNN Algorithm*. <https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761>. Sept. 2018.
- [9] E.; Kalech Khalastchi. "Fault Detection and Diagnosis in Multi-Robot Systems: A Survey". In: ().
- [10] James Kramer and Matthias Scheutz. "Development environments for autonomous mobile robots: A survey". In: *Autonomous Robots* 22.12 (Feb. 2007), pp. 1573–7527. URL: <https://doi.org/10.1007/s10514-006-9013-8>.
- [11] Anastassia Kuestenmacher and Paul G. Plöger. *Model-Based Fault Diagnosis Techniques for Mobile Robots*. 2016. DOI: 10.1016/j.ifacol.2016.07.613.
- [12] Anastassia Kuestenmacher and Paul G. Plöger. "Model-Based Fault Diagnosis Techniques for Mobile Robots**This work was sponsored by the B-IT foundation and the Strukturfond des Landes Nordrhein-Westfalen for the female PhD students." In: *IFAC-PapersOnLine* 49.15 (2016). 9th IFAC Symposium on Intelligent Autonomous Vehicles IAV 2016, pp. 50–56. ISSN: 2405-8963. DOI: <https://doi.org/10.1016/j.ifacol.2016.07.613>. URL: <http://www.sciencedirect.com/science/article/pii/S2405896316308849>.
- [13] M. Luo et al. "Model-based fault diagnosis/prognosis for wheeled mobile robots: a review". In: *31st Annual Conference of IEEE Industrial Electronics Society, 2005. IECON 2005*. 2005, 6 pp.-.
- [14] Sarang Narkhede. *AUC - ROC Curve*. <https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5>. June 2018.
- [15] Ola Pettersson, Lars Karlsson, and Alessandro Saffiotti. "Model-Free Execution Monitoring in Behavior-Based Robotics". In: *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man,*

- and Cybernetics Society* 37 (Sept. 2007), pp. 890–901. DOI: 10.1109/TSMCB.2007.895359.
- [16] Morgan Quigley, Eric Berger, and Andrew Y. Ng. “STAIR: Hardware and Software Architecture”. In: 2007.
- [17] Dr.Kashif Qureshi. *Cutting-Edge Evolutions of Information Technology. Artificial intelligence & Machine Learning*. Booksclinic Publishing, 2019.
- [18] V. Verma et al. “Real-time fault diagnosis [robot fault diagnosis]”. In: *IEEE Robotics Automation Magazine* 11.2 (2004), pp. 56–66.
- [19] K. A. Wyrobek et al. “Towards a personal robotics development platform: Rationale and design of an intrinsically safe personal robot”. In: *2008 IEEE International Conference on Robotics and Automation*. May 2008, pp. 2165–2170. DOI: 10.1109/ROBOT.2008.4543527.

Appendix A

A.1 Data streams

When powered on, kobuki robot sends byte streams to the driver software through serial communication at 50 Hz frequency. The byte streams of the sensors used in this project are :

- Raw data from 3-axis Digital Gyroscope which is later converted in to angular velocities and angular accelerations by Deserialization and sent to ROS system as imu data.

	Name		Size	Value	Value in Hex	Description
Header	Identifier		1	13	0x0D	Fixed
Length	Size of data field		1	2+6N		
Data	Frame id		1			Frame id of 'Raw gyro data 0' Every sensor readings can identified by frame id , It circulate from 0 to 255
	Followed data length		1	3N		
	Raw gyro data 0	x-axis	2			ADC output of each-axis in 0.00875 deg/s
		y-axis	2			
		z-axis	2			
	:					
	:					
	Raw gyro data N-1	x-axis	2			
		y-axis	2			
		z-axis	2			

Figure A.1: IMU byte stream

- Current sensor readings of wheel motors.

	Name	Size	Value	Value in Hex	Description
Header	Identifier	1	6	0x06	Fixed
Length	Size of data field	1	2	0x02	Fixed
Data	Left motor	2			in 10mA
	Right motor	2			in 10mA

Figure A.2: Current byte stream

- Data of wheel drop sensor, Bumper sensor and wheel encoder values are sent together as part of basic core sensor data byte stream.

	Name	Size	Value	Value in Hex	Description
Header	Feedback Identifier	1	1	0x01	Fixed
Length	Size of data field	1	15	0x0F	Fixed
Data	Timestamp	2			Timestamp generated internally in milliseconds It circulates from 0 to 65535
	Bumper	1			Flag will be setted when bumper is pressed 0x01 for right bumper 0x02 for central bumper 0x04 for left bumper
	Wheel drop	1			Flag will be setted when wheel is dropped 0x01 for right wheel 0x02 for left wheel
	Cliff	1			Flag will be setted when cliff is detected 0x01 for right cliff sensor 0x02 for central cliff sensor 0x04 for left cliff sensor
	Left encoder	2			Accumulated encoder data of left and right wheels in ticks Increments of this value means forward direction It circulates from 0 to 65535
	Right encoder	2			
	Left PWM	1			PWM value that applied to left and right wheel motor This data should be converted signed type to represent correctly Negative sign indicates backward direction
	Right PWM	1			
	Button	1			Flag will be setted when button is pressed 0x01 for Button 0 0x02 for Button 1 0x04 for Button 2
	Charger	1			0 for DISCHARGING state 2 for DOCKING_CHARGED state 6 for DOCKING_CHARGING state 18 for ADAPTER_CHARGED state 22 for ADAPTER_CHARGING state
	Battery	1			Voltage of battery in 0.1 V Typically 16.7 V when fully charged
	Overcurrent flags	1			Flag will be setted when overcurrent is detected 0x01 for left wheel 0x02 for right wheel

Figure A.3: Basic Core Sensor Data

Appendix B

B.1 Implementation

Launch file for launching required nodes is

```
from launch import LaunchDescription
import launch_ros.actions

def generate_launch_description():
    return LaunchDescription([
        launch_ros.actions.Node(
            package='turtlebot2_drivers', node_executable='kobuki_node', output='screen'
        ),
        launch_ros.actions.Node(
            package='robot_nav', node_executable='navigation_node', output='screen'
        ),
    ])
)
```

Figure B.1: Launch file

B.1.1 Visualization

The extracted data from kobuki robot for each use case scenario is visualized using python plots. The feature values of currents, bumper and wheel drop are plotted against time. The plots for the use case are :

(i) No obstacle :

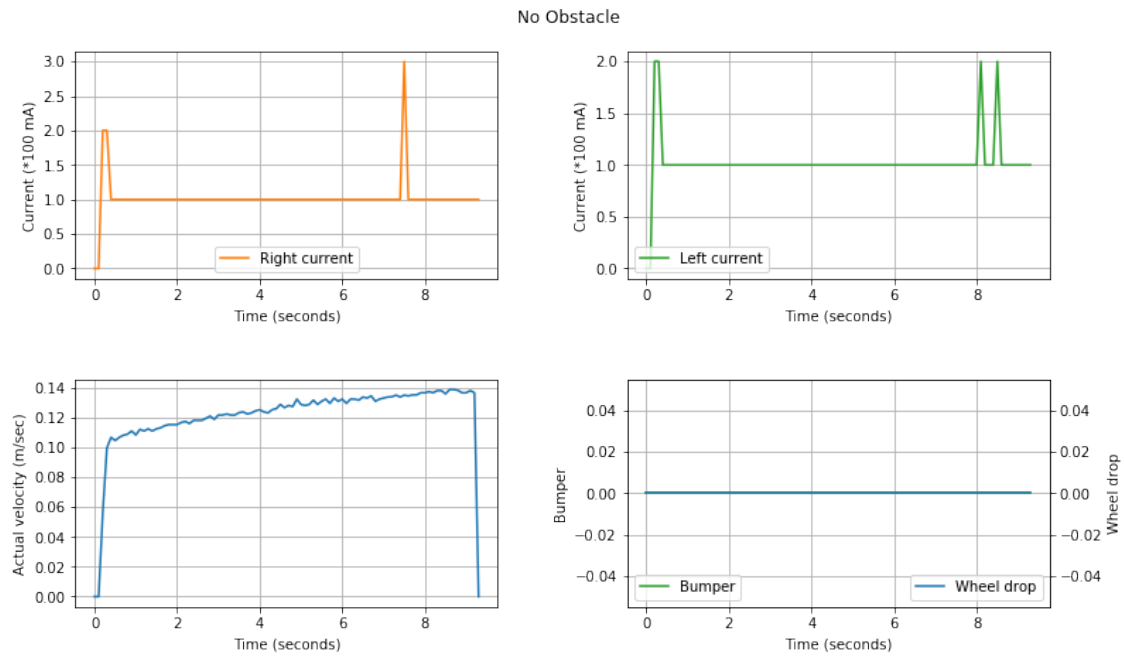


Figure B.2: No obstacle

(ii) Rigid Obstacle :

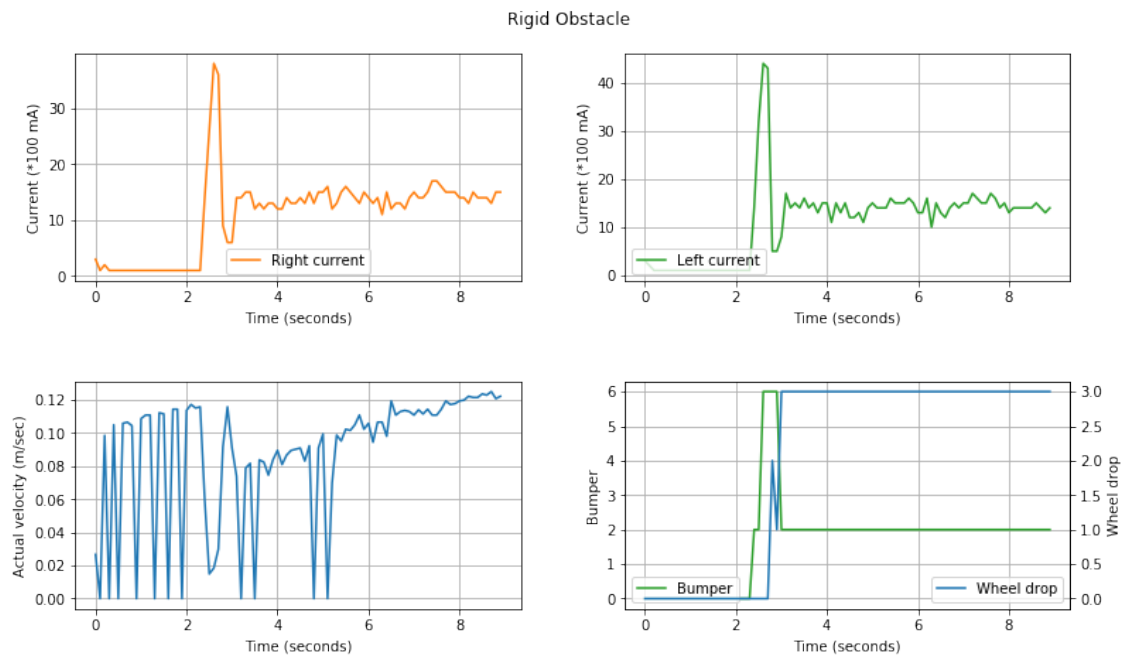


Figure B.3: Rigid obstacle

(iii) Movable Obstacle :

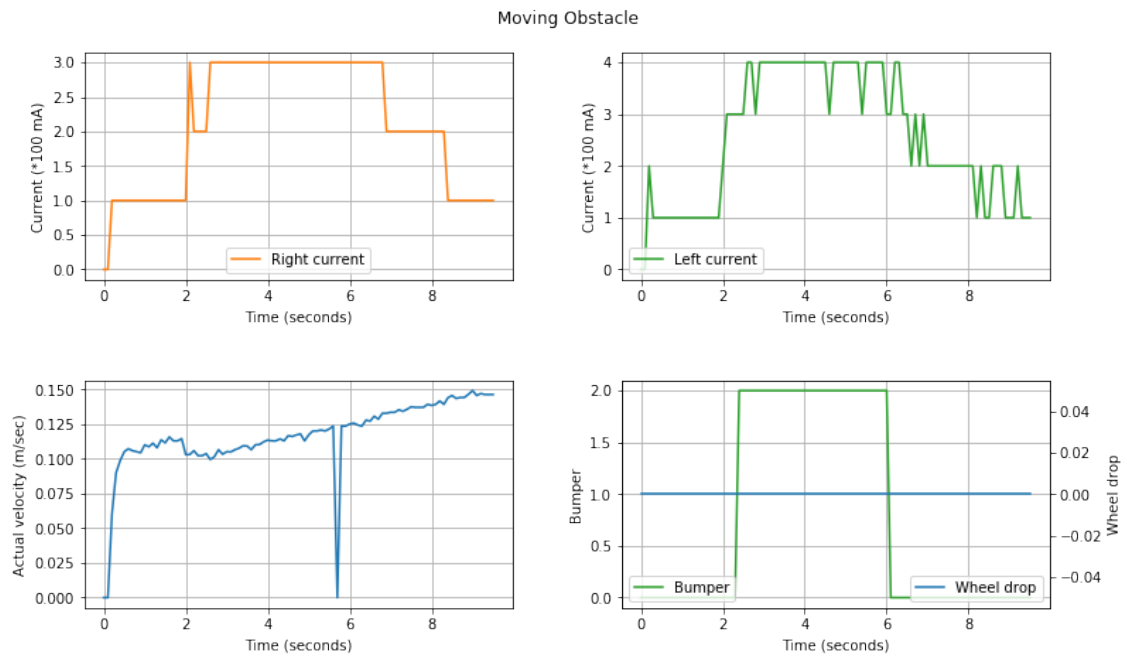


Figure B.4: Movable obstacle

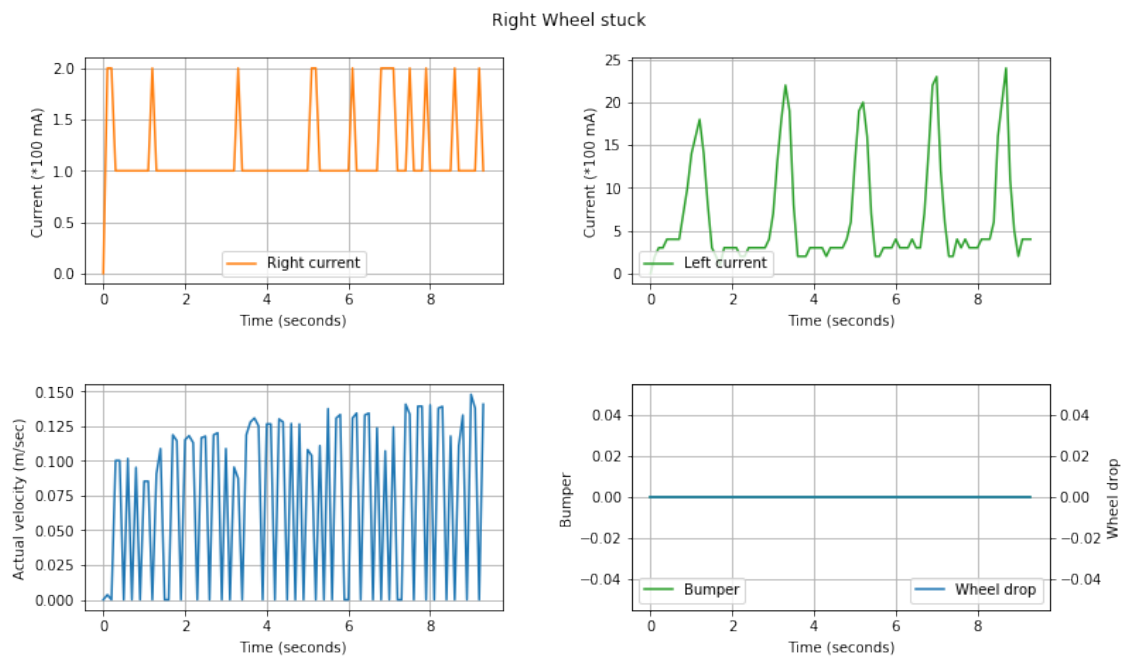
(iv) **Wheel stuck :**

Figure B.5: Stuck wheels

(v) **Slip :**

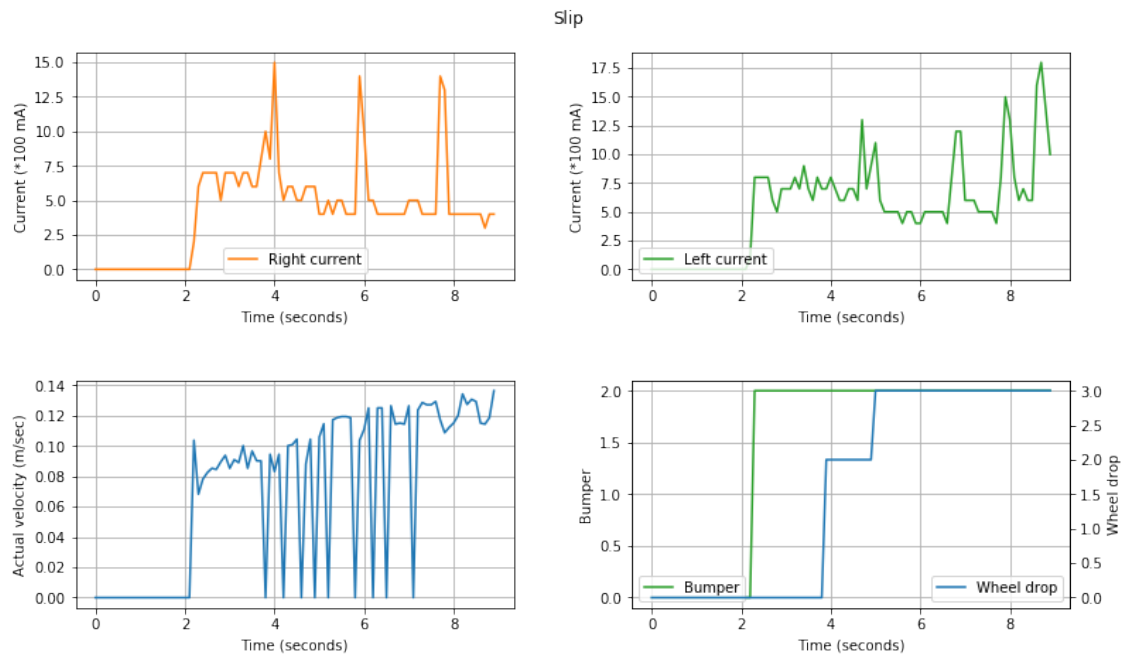


Figure B.6: Slip

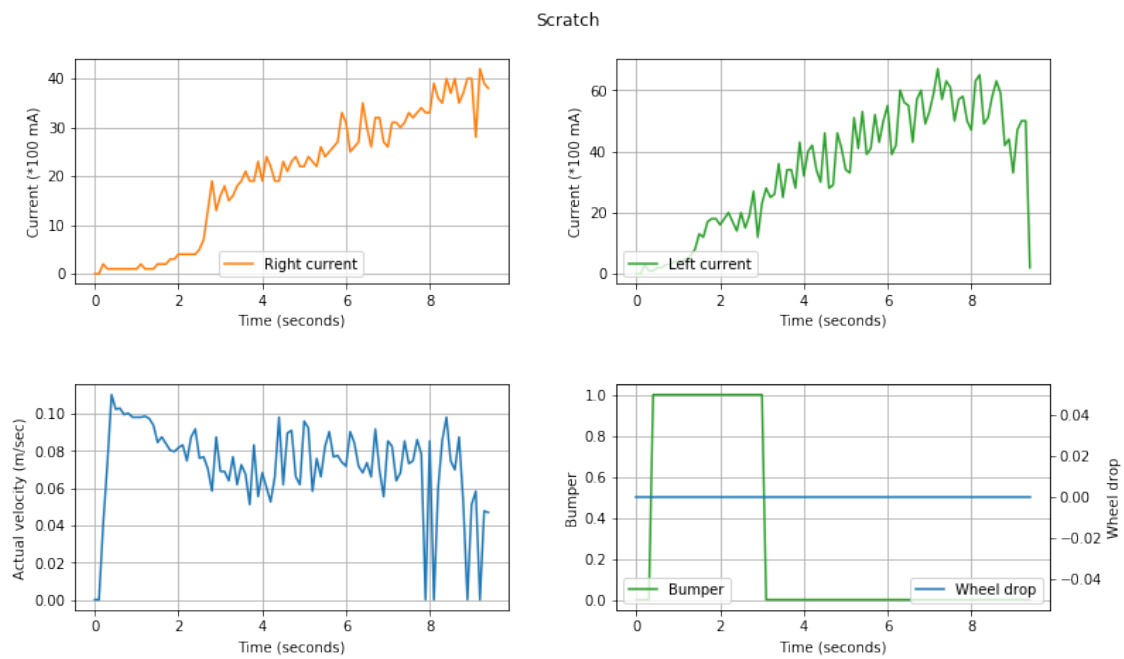
(vi) **Scratch :**

Figure B.7: Slip

(vii) **Wedge :**

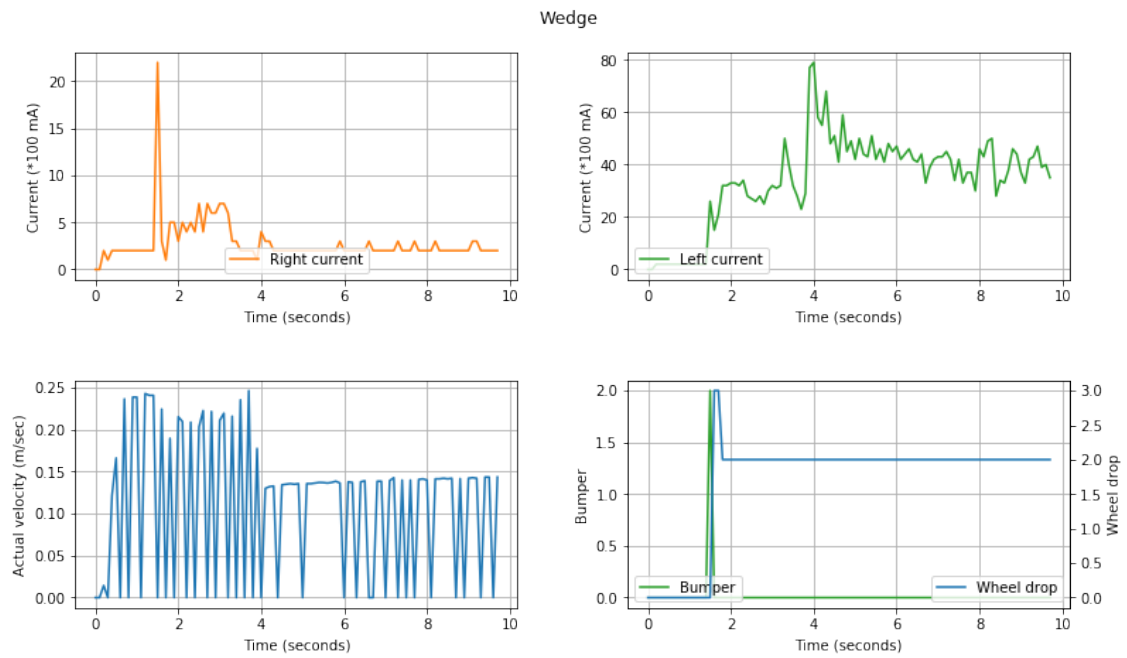


Figure B.8: Wedge

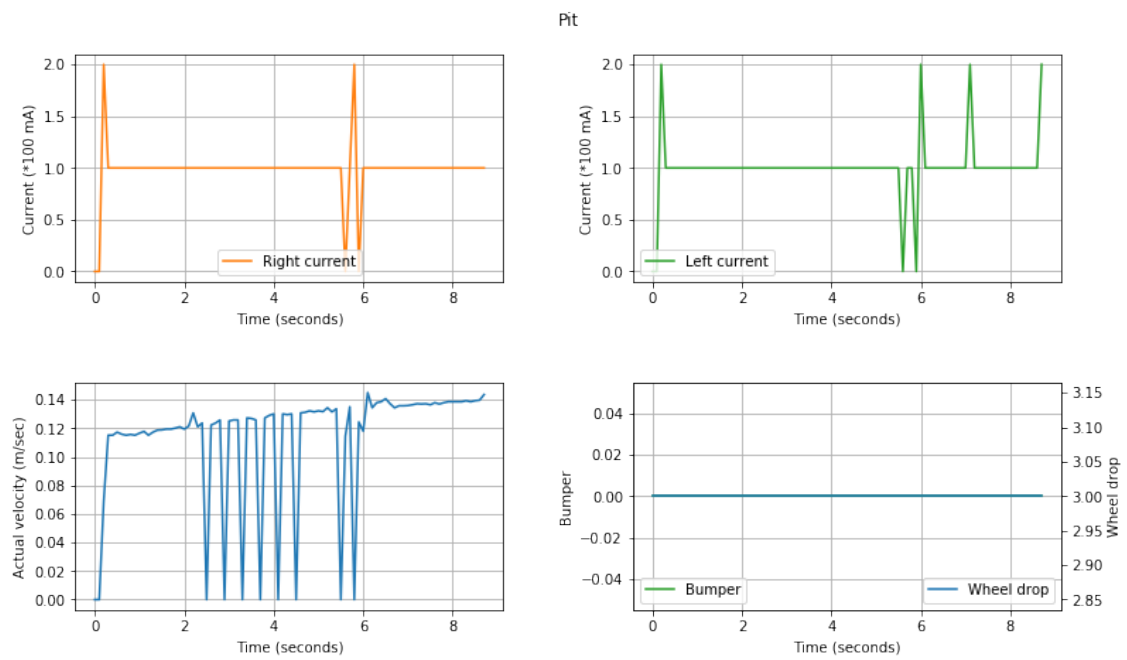
(viii) **Pit :**

Figure B.9: Pit

B.1.2 Machine Learning

All the required packages are imported to the programming environment Jupyter Notebook

```
import pandas as pd
import numpy as np
from mlxtend.plotting import plot_decision_regions
import matplotlib.pyplot as plt
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from pandas.plotting import scatter_matrix
import seaborn as sns
sns.set()
```

Figure B.10: Importing packages

Data is split in to training and test datasets and scaled to a zero mean , unit variance.

```
#Obtain features and labels
X = dataset.iloc[:, 0:7]
y = dataset.iloc[:, 7]
#Split the data
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=0, test_size=0.3)
#Scale the data
scale = StandardScaler()
X_train = scale.fit_transform(X_train)
X_test = scale.transform(X_test)
```

Figure B.11: formatting data

A Knn classification model is defined and trained (fit) using the training dataset and then tested using test dataset to obtain y_pred and accuracy metrics are obtained.

```
#Defining the model
Model = KNeighborsClassifier(n_neighbors=4,p=2,metric= 'euclidean')
#Training the model
Model.fit(X_train, y_train)
#Testing the model
y_pred =Model.predict(X_test)
```

Figure B.12: Training