

LAB MANUAL

Deep Learning for Robotics

Experiment 1: Building And Programming of Teleoperation Differential Drive Robot Virtually

Aim:

Construct a differential robot using primitive shapes and program it to demonstrate fundamental movements

Description:

Differential Drive Robot: A differential drive robot is a type of mobile robot that moves by varying the speeds of its two drive wheels. Here's how it works:

Drive System: The robot is typically driven by two wheels mounted on either side of the robot's body. Each wheel can be independently controlled, allowing the robot to move forward, backward, or turn in place.

Differential Steering: To move forward, both wheels rotate in the same direction at the same speed. To turn, the wheels rotate in opposite directions, with the wheel on the side of the turn moving slower than the wheel on the other side.

Speed Control: By controlling the speed of each wheel independently, the robot can achieve different types of movements, such as straight-line motion, curves, and spins.

Limitations: Differential drive robots can be prone to slipping, especially when turning on uneven or slippery surfaces. This can affect the accuracy of their movements.

Differential drive robots are commonly used in robotics education and research due to their simplicity and effectiveness in a wide range of applications. They are often used in tasks such as exploration, mapping, and surveillance.

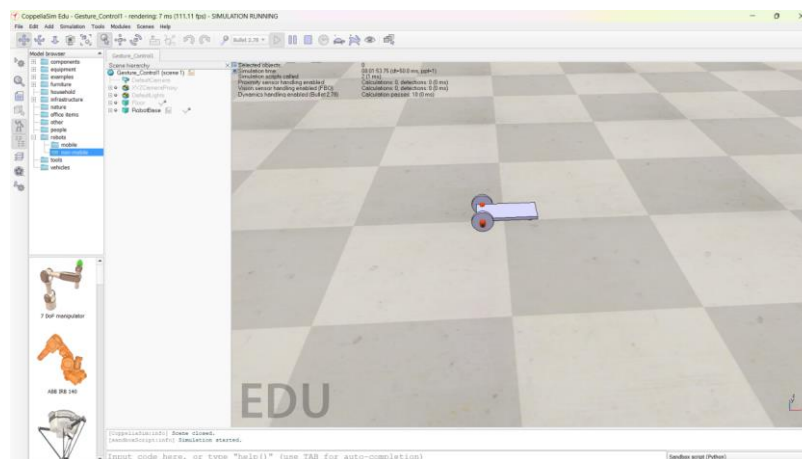
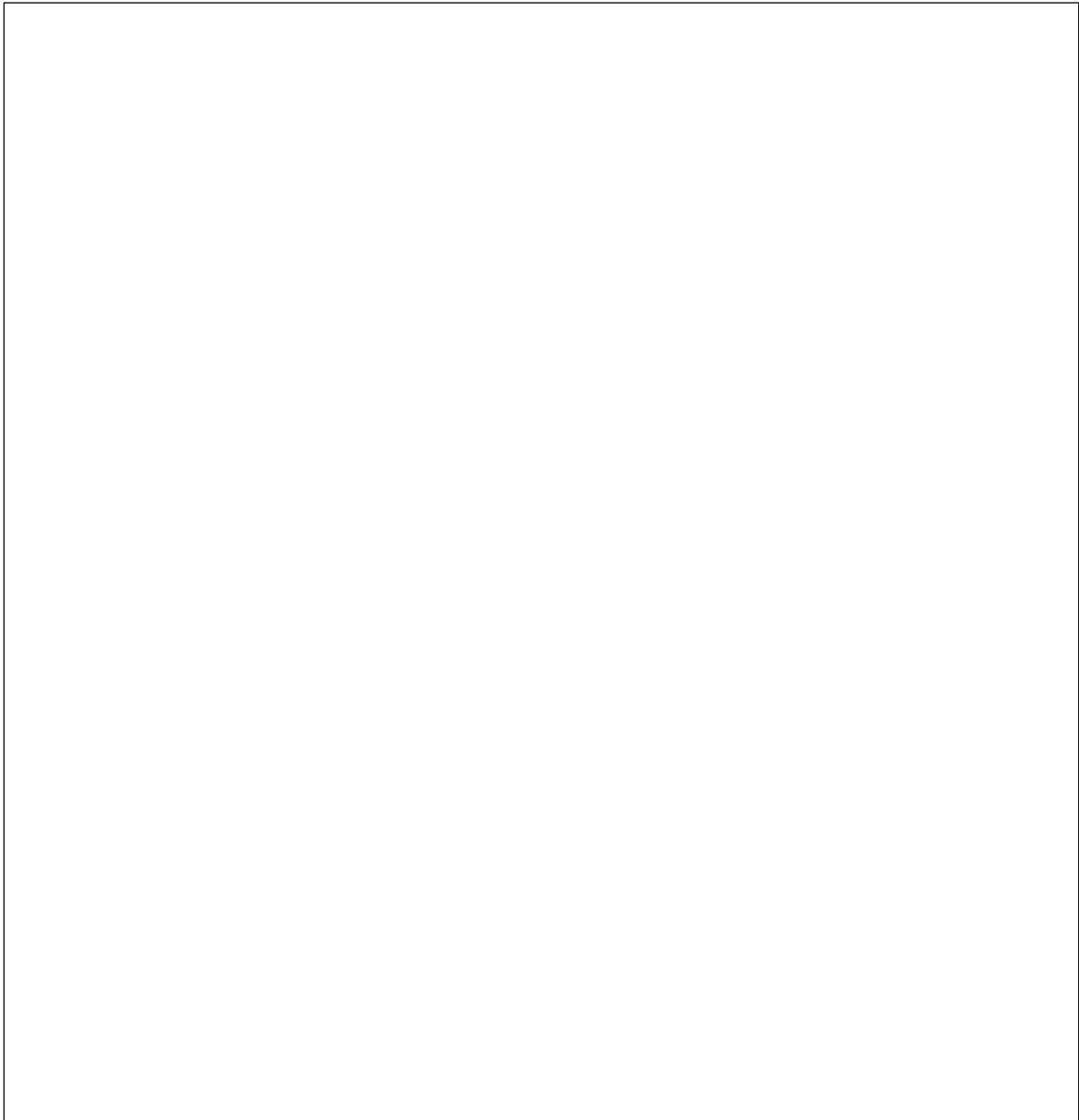


Image Source - Differential Drive Robot using primitive shapes in Coppeliasim

Algorithm:

- Import the required primitive shapes of size needed for the robot
- Add the revolute joint to the design as these are going to act as motors for the robot
- Build and position the caster wheel along with a force sensor
- Provide parent-child relationship to the components we have added
- Create a script for the built robot
- Open the script and initialize the motors
- Build the logic for the key press.
- Program it in LUA for the basic movements of the robot
- Play and check the simulation

Code Implementation:

Result: The construction of the differential drive robot is completed, and the teleoperation via keyboard key press is programmed and validated through simulation in the environment.

Experiment 2: Fundamentals Of Arduino

Aim: To grasp the fundamentals of Arduino, program it using the IDE, and upload the code to verify its functionality.

Requirements:

- Arduino UNO Board
- USB Cable
- Arduino IDE Software

Explanation

Arduino: Arduino is an open-source electronics platform based on easy-to-use hardware and software.

Here are its key fundamentals:

Microcontroller: Arduino boards are equipped with microcontrollers (usually from the Atmel AVR or ARM Cortex families) that act as the brains of the board. These microcontrollers can be programmed to perform a variety of tasks.

Arduino IDE: The Arduino Integrated Development Environment (IDE) is a software application used to write and upload code to the Arduino board. It provides a simple interface for writing code in the Arduino programming language, which is based on C and C++.

Input/Output Pins: Arduino boards have a number of digital and analog input/output pins that can be used to interface with external sensors, actuators, and other electronic components. These pins can be configured as inputs or outputs in software.

Power Supply: Arduino boards can be powered via USB, a DC power jack, or an external power source. They typically operate at 5V, but some boards also support 3.3V operation.

Expansion Shields: Arduino boards can be expanded using shields, which are additional circuit boards that plug into the main Arduino board. Shields can add functionality such as wireless communication, motor control, and sensor interfaces.

Programming Language: Arduino uses its own programming language, which is similar to C and C++. It provides a set of libraries and functions that simplify common tasks such as reading analog sensors, controlling servos, and communicating over serial.

Overall, Arduino is a versatile platform that can be used for a wide range of projects, from simple blinking LED experiments to complex robotics and automation applications. Its ease of use, affordability, and large ecosystem of compatible hardware and software make it a popular choice for hobbyists, students, and professionals alike.

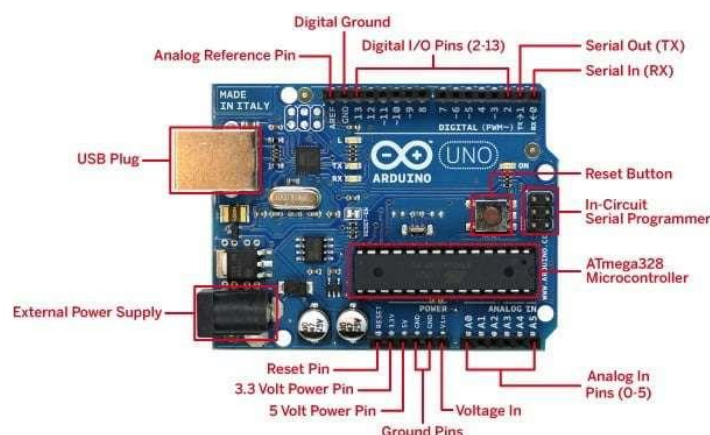
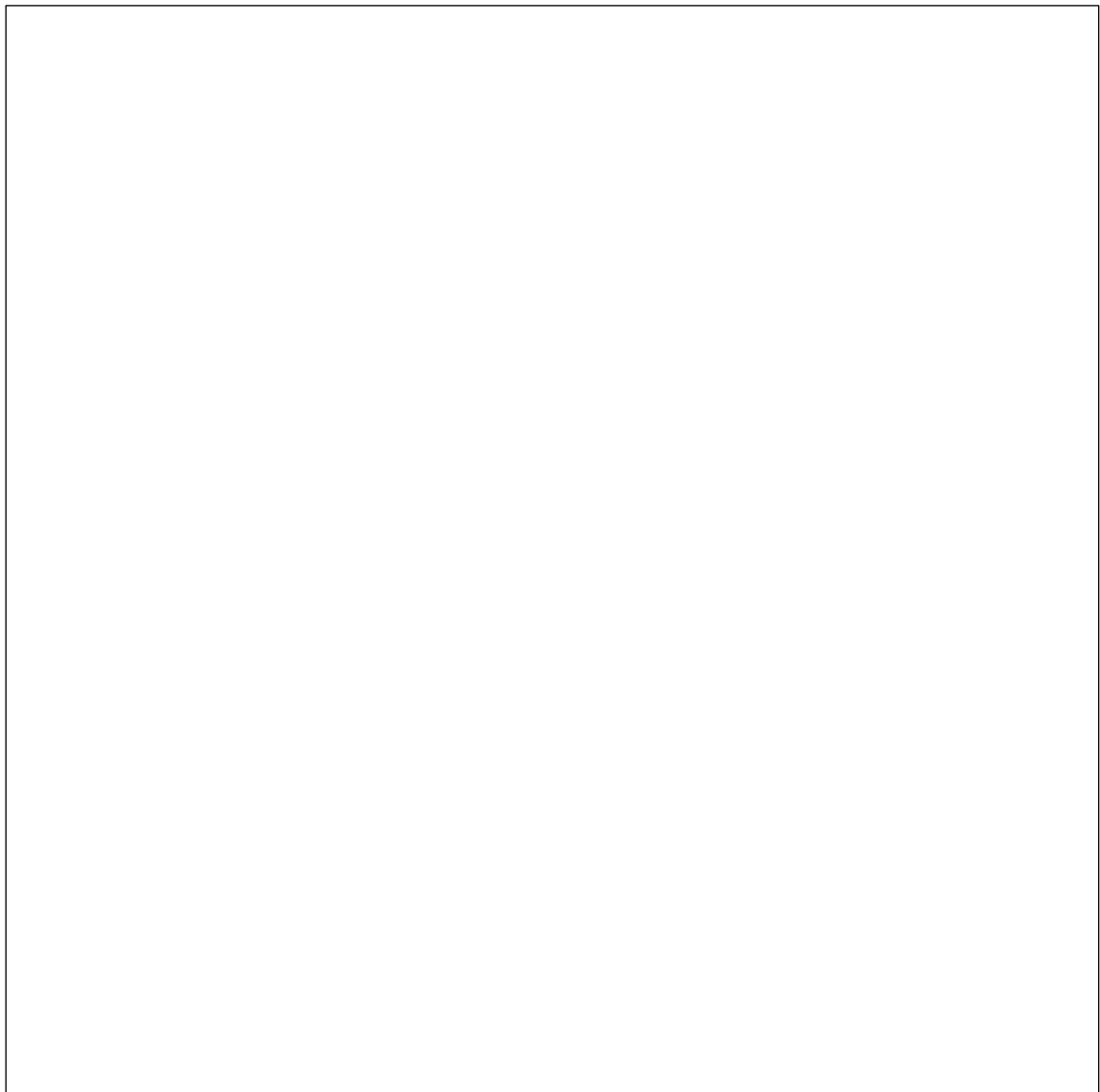


Image Source - <https://robosynckits.in/wp-content/uploads/2020/12/arduino-uno-r3-pic3.j>

Steps

- Download and install the Arduino IDE
- Get a new file
- Initialize the in-built LED
- Program the logic such that it blinks for one second
- Connect the Arduino UNO board to the computer
- Select the appropriate board and its corresponding port
- Upload the program
- Once upload is complete the in-built LED will blink

Code Implementation:

Result: Therefore, the fundamental aspects of Arduino have been learned, and a basic program to blink the built-in LED has been developed, tested, and uploaded to the board for visualization.

Experiment: 3 Building A Bluetooth Controlled Robot

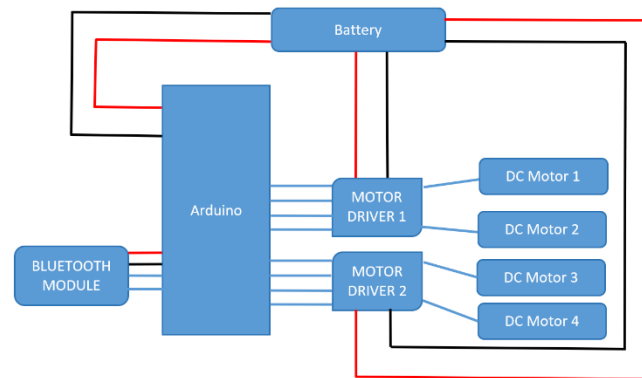
Aim:

To construct and evaluate a wireless robot control system using Bluetooth.

REQUIREMENTS:

- Arduino UNO Board
- USB Cable
- DC Motors
- Motor Driver
- HC05 BT Module
- LiPo Battery
- Jumper Wires
- Robot Chassis
- Required Hardware
- Arduino IDE Software

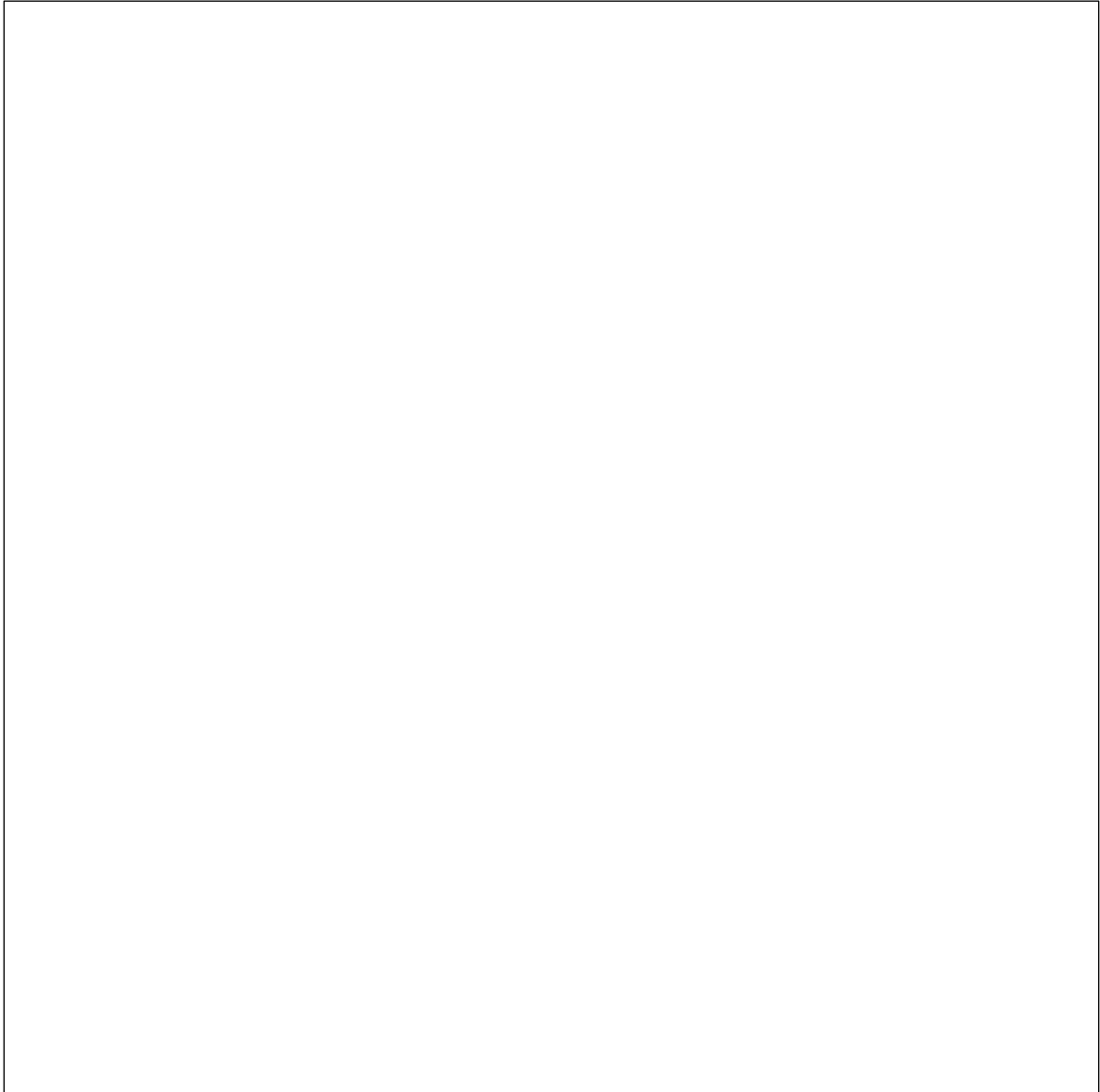
CONNECTION DIAGRAM



Steps

- Build the robot using robot chassis and hardware
- Place the electronic components and wire them using jumper wires
- Get a new file
- Initialize the motors and serial begin the communication module
- Program the logic such that for every running number we have a set of motion to be executed
- Connect the Arduino UNO board to the computer
- Select the appropriate board and its corresponding port
- Upload the program
- Once uploaded, connect the battery to turn on
- In mobile get an open-source Arduino Bluetooth connector app
- Through the app connect to the robot Bluetooth
- Now provide the number and corresponding motion is executed from program

Code Implementation:



Result: Therefore, the robot is wireless controlled via Bluetooth, and a set of options are executed by sending corresponding numbers through a mobile app.

Experiment No:4 Building an Obstacle Avoidance Robot

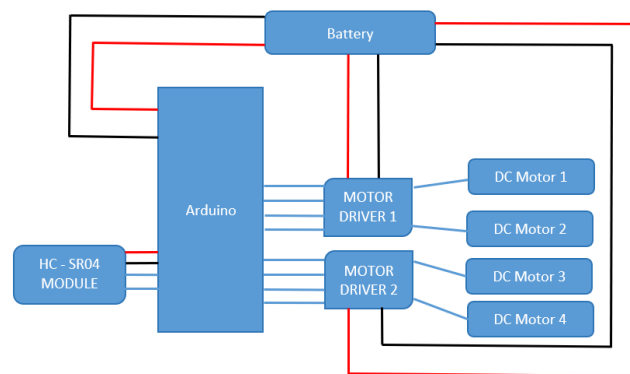
Aim:

To construct and program the robot to navigate an environment without collisions.

Requirements:

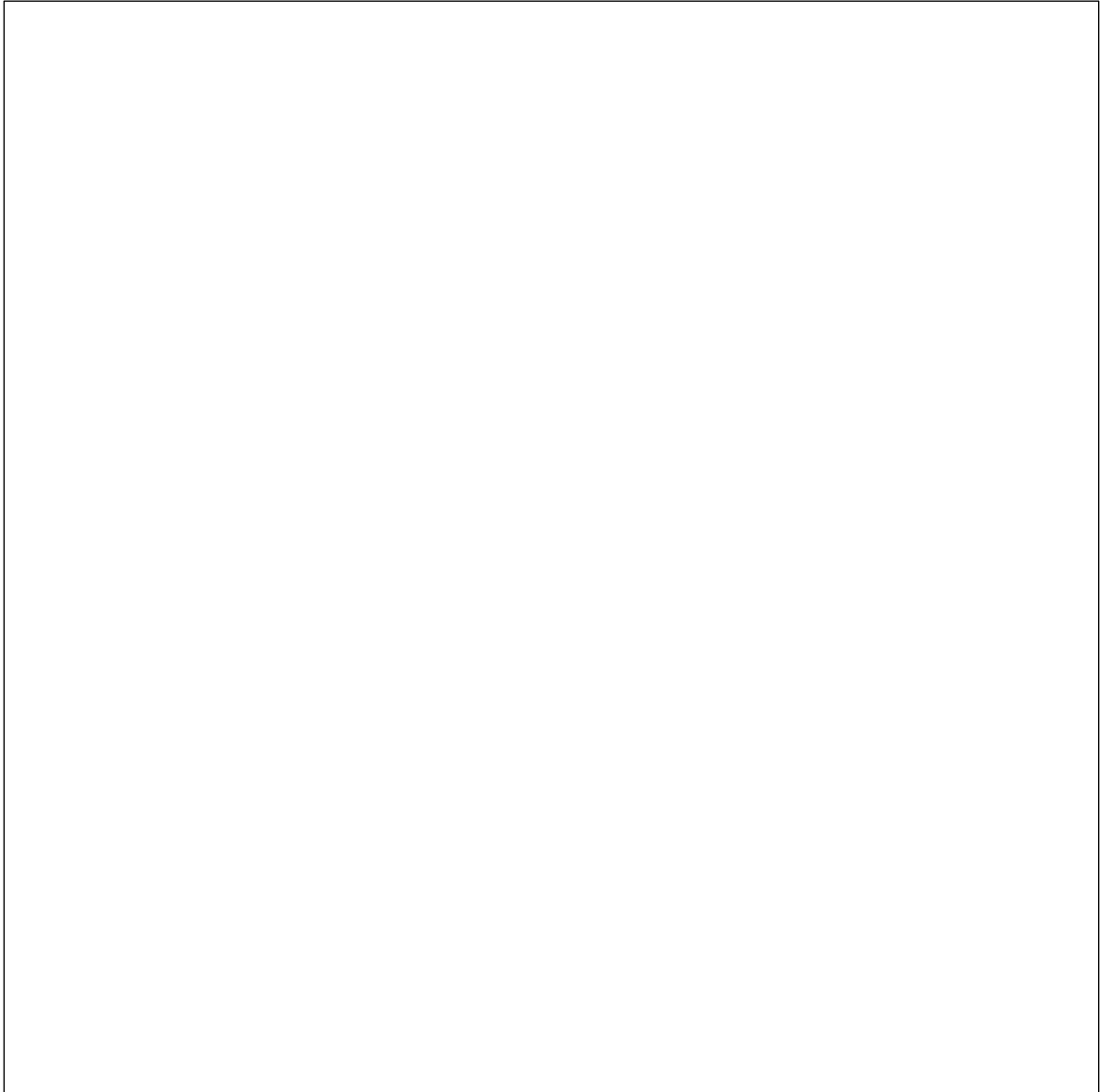
- Arduino UNO Board
- USB Cable
- DC Motors
- Motor Driver
- HC-SR04 Module
- LiPo Battery
- Jumper Wires
- Robot Chassis
- Required Hardware
- Arduino IDE Software

Connection Diagram



Steps

- Build the robot using robot chassis and hardware
- Place the electronic components and wire them using jumper wires
- Get a new file
- Program the logic such that when a certain distance is reached from the sensor the robot has to stop or take a right turn.
- Once no obstacle is present the robot has to move in the way
- Connect the Arduino UNO board to the computer
- Select the appropriate board and its corresponding port
- Upload the program
- Once uploaded, connect the battery to turn on
- Now the robot moves and if a hand or object is kept in front of sensor and gets detected then the stop or right turn will happen
- The system is continuing and will stop only when the robot is switched off

Code Implementation:**Result:**

The robot is constructed with sensors and programmed to detect obstacles when they reach a certain distance. Finally, the robot is executed to move around without collisions.

Experiment No:5 Build A Simple Neural Network

Aim:

Constructing a simple neural network in TensorFlow to execute logistic regression and categorize the output.

Description:

Logistic regression is a statistical method used for binary classification tasks, where the goal is to predict the probability that an instance belongs to a particular class (usually denoted as class 1). Here's an explanation of logistic regression and how it's built in TensorFlow 2

Logistic Regression:

Model:

- In logistic regression, the output of the model is obtained by applying the logistic function (also known as the sigmoid function) to a linear combination of the input features:

$$= (w_1x_1 + w_2x_2 + \dots + w_nx_n + b)$$

- Here, y represents the predicted probability that the instance belongs to class 1, x_1, x_2, \dots, x_n are the input features, w_1, w_2, \dots, w_n are the weights, b is the bias term, and σ is the sigmoid function.

Sigmoid Function:

- The sigmoid function transforms the linear combination of inputs into a value between 0 and 1, representing the probability of belonging to class 1:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Training:

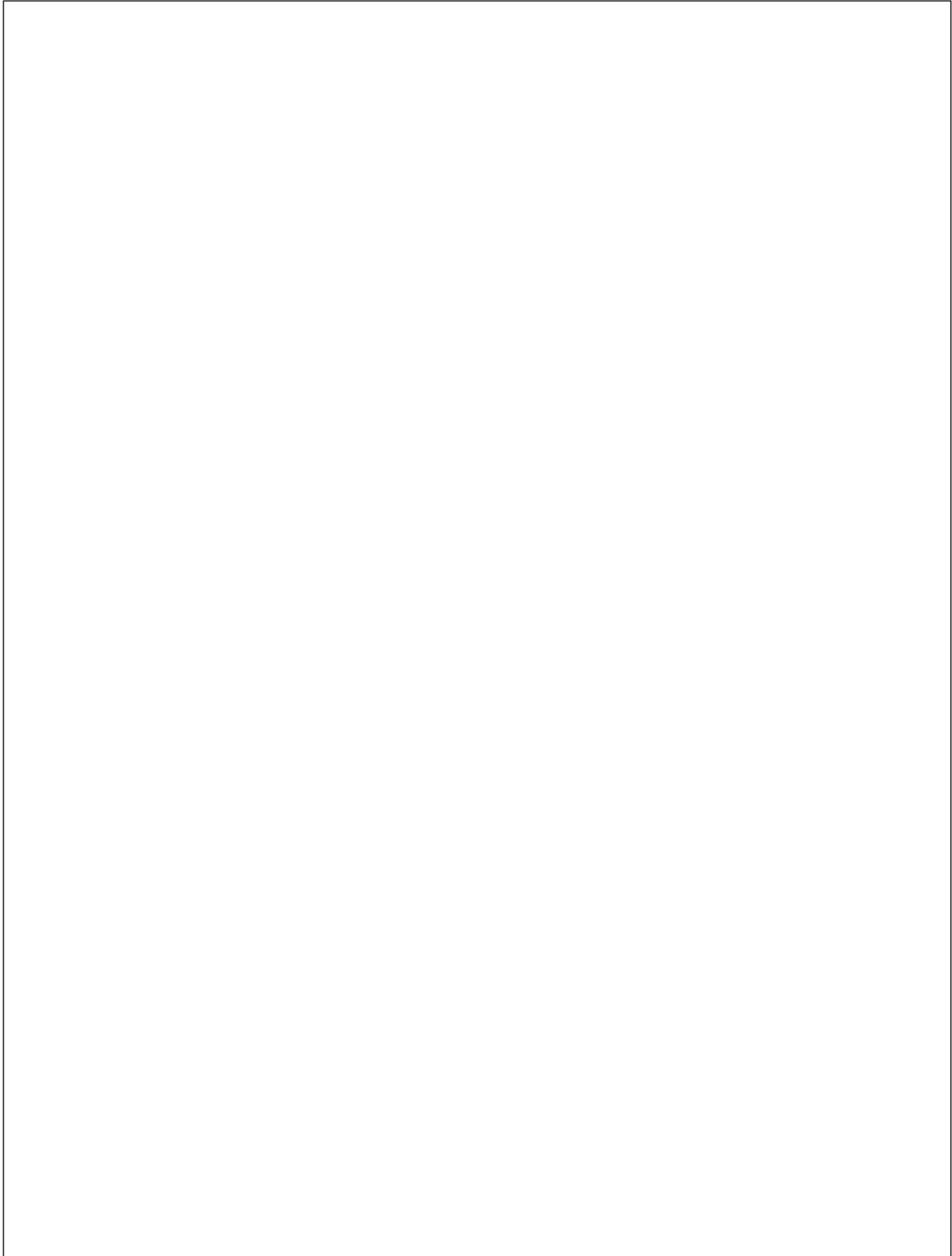
- During training, the model learns the optimal values of weights w and bias b by minimizing a loss function (e.g., binary cross-entropy loss) using optimization algorithms like gradient descent.

Prediction:

- Once trained, the model can predict the probability of belonging to class 1 for new instances by passing their features through the trained logistic regression model.

Algorithm

- Import necessary libraries and the dataset
- Create a TensorFlow sequential model and a sigmoid activation function
- Compile the model with appropriate optimizer and a suitable loss function
- Train the model using fit method by providing input and target
- Specify the number of epochs for training
- Predict using new data and evaluate the model's performance

Code Implementation:**Result:**

Thus, utilizing TensorFlow, a simple neural network is constructed to implement logistic regression with a dataset and classify the output.

Experiment No:6 Build A Multi-Layer Perceptron Model

Aim:

To build a simple single layer perceptron model in python using TensorFlow 2

Explanation:

Multi-Layer Perceptron

A multi-layer perceptron (MLP) is a type of feedforward artificial neural network that consists of multiple layers of nodes, each connected to the next in a sequential manner. MLPs are used for supervised learning tasks, including classification and regression.

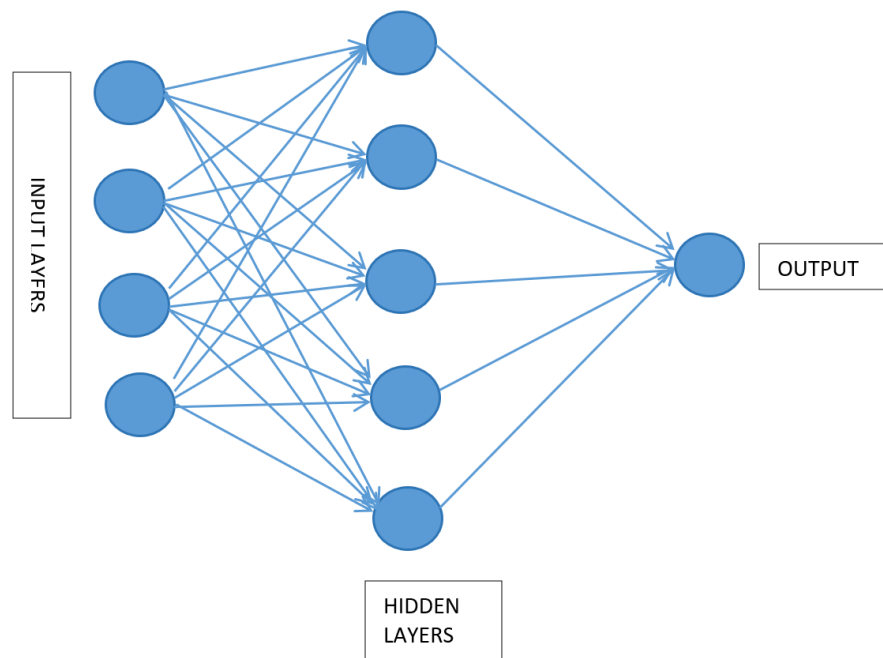
Input Layer: The input layer receives the features of the input data.

Hidden Layers: One or more hidden layers process the input data through a series of weighted connections and activation functions. Each node in a hidden layer takes the weighted sum of its inputs, applies an activation function (such as ReLU or sigmoid), and passes the result to the next layer.

Output Layer: The output layer produces the final predictions or outputs of the model. The number of nodes in the output layer depends on the task (e.g., binary classification, multi-class classification, regression).

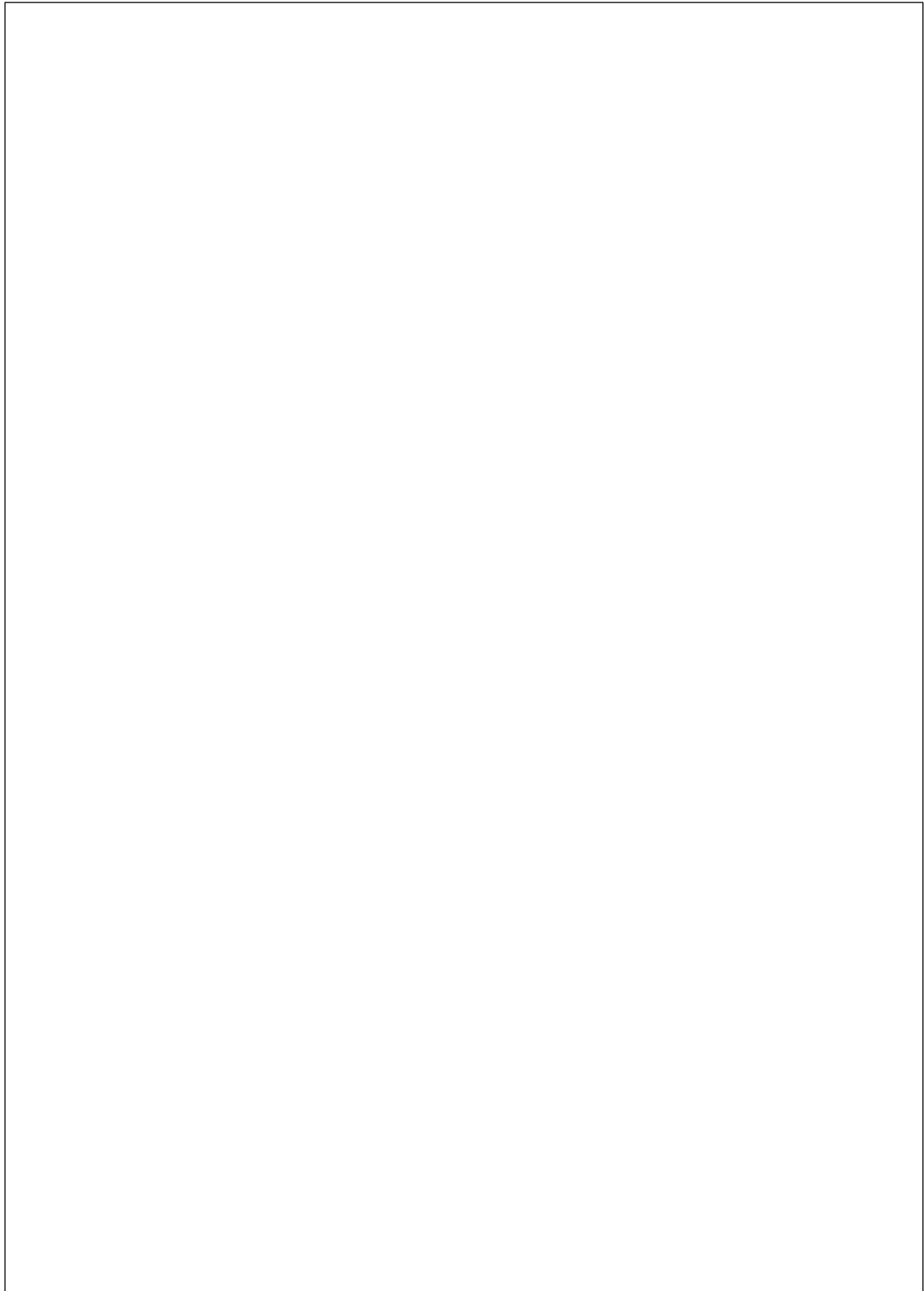
Activation Functions: Activation functions introduce non-linearity into the model, allowing it to learn complex patterns in the data. Common activation functions include ReLU, sigmoid, and tanh.

Training: MLPs are trained using backpropagation, where the error between the predicted output and the actual output is calculated (using a loss function) and propagated back through the network to update the weights using an optimization algorithm (such as SGD or Adam).



Algorithm:

- Import dataset
- Create the network architecture
- Build and compile the neural network
- Fit the values and train with set number of epochs and in batch.
- Predict the outcome with new values

Code Implementation:**Result:**

A rudimentary multi-layer perceptron model is established with the Iris dataset, after which its evaluation score is computed.

Experiment No :7 Optimization Using Stochastic Gradient Descent Algorithm

Aim:

Implementing a stochastic gradient descent algorithm for optimization.

Description:

Stochastic gradient descent (SGD) is an optimization algorithm commonly used for training machine learning models. Unlike batch gradient descent, which computes the gradient of the loss function using the entire dataset, SGD computes the gradient using a single randomly chosen data point (or a small subset of points) at each iteration. Here's how SGD works:

Initialization: Start by initializing the model parameters (weights and biases) to some random values.

Choose a Learning Rate: Select a learning rate (α), which controls the size of the step taken in the direction of the gradient.

Iterate Until Convergence: Repeat the following steps until the algorithm converges:

- Randomly shuffle the training data.
- For each data point $(x^{(i)}, y^{(i)})$ in the training data:
 - Compute the gradient of the loss function with respect to the parameters using only that data point:

$$\nabla J(\theta; x^{(i)}, y^{(i)})$$

- Update the parameters using the computed gradient:

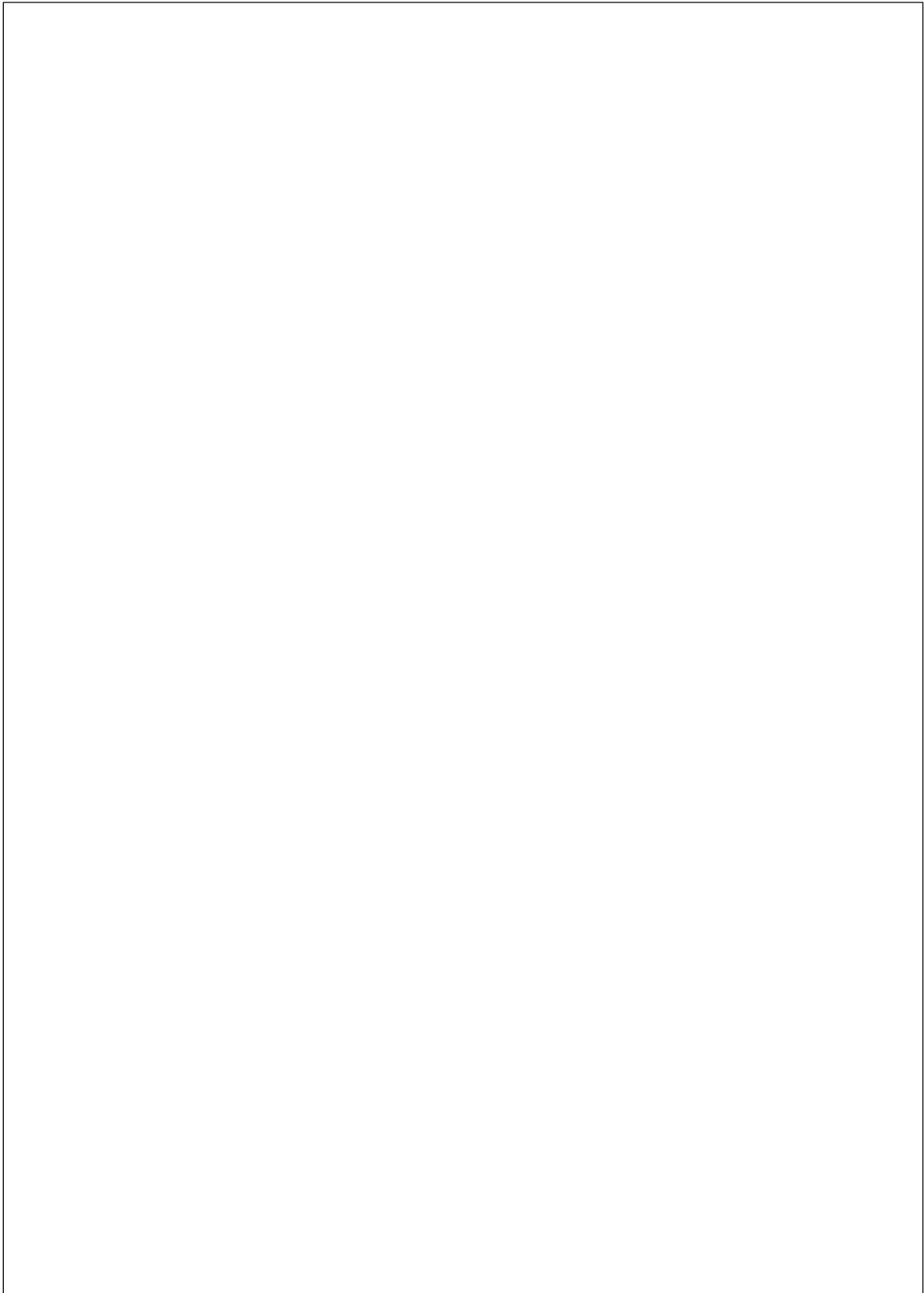
$$\theta = \theta - \alpha \cdot \nabla J(\theta; x^{(i)}, y^{(i)})$$

Stopping Criterion: The algorithm stops when a stopping criterion is met, such as reaching a maximum number of iterations or when the change in the parameters is below a certain threshold.

SGD is particularly useful when working with large datasets, as it can be computationally more efficient than batch gradient descent. However, it can also be more sensitive to the learning rate and may require tuning of hyperparameters to achieve good performance. In practice, a variant of SGD called mini-batch gradient descent is often used, where the gradient is computed using a small batch of data points rather than a single point. This helps to reduce the variance in parameter updates while still being more computationally efficient than batch gradient descent.

Algorithm:

- Initialize parameters
- Define model
- Set num of iterations, penalty
- Fit the values
- Get values of weights, intercept

Code Implementation:**Result:**

Thus, a simple implementation of stochastic gradient descent is completed and executed.

Experiment No :8 BUILD AN IMAGE RECOGNITION SYSTEM USING CNN

Aim

Constructing layers from the ground up to create a CNN model for image recognition using TensorFlow 2

Description:

CNN Model

A Convolutional Neural Network (CNN) is a type of deep learning model that is particularly effective for image recognition and classification tasks. CNNs are inspired by the organization of the animal visual cortex, and they have proven to be very successful in various computer vision tasks. Here's how CNNs work and how they are built in TensorFlow 2:

Convolutional Layer:

- The core building block of a CNN is the convolutional layer.
- Each convolutional layer applies a set of filters (also called kernels) to the input image.
- These filters detect different features of the input, such as edges, textures, and shapes.
- The output of a convolutional layer is a set of feature maps, each representing the result of applying one filter to the input image.

Activation Function:

- After each convolution operation, an activation function like ReLU (Rectified Linear Unit) is applied element-wise to introduce non-linearity into the network.

Pooling Layers:

- Pooling layers are used to down sample the feature maps, reducing their spatial dimensions.
- Common pooling operations include max pooling and average pooling, which take the maximum or average value over a small region of the feature map, respectively.

Flattening:

- After several convolution and pooling layers, the remaining feature maps are flattened into a one-dimensional vector.

Fully Connected Layers:

- The flattened vector is passed through one or more fully connected layers, which perform classification based on the learned features.
- The last fully connected layer typically has nodes corresponding to the number of classes in the classification task.

Output Layer:

- The final layer produces the output of the network, representing the predicted class probabilities for classification tasks.

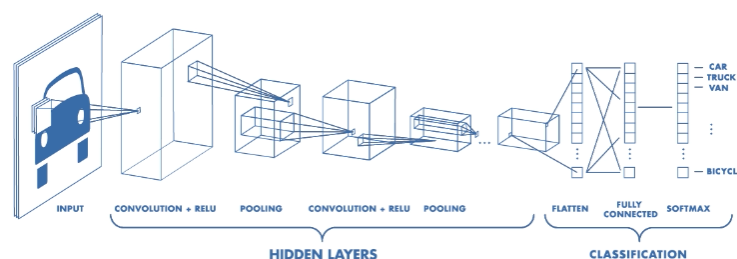
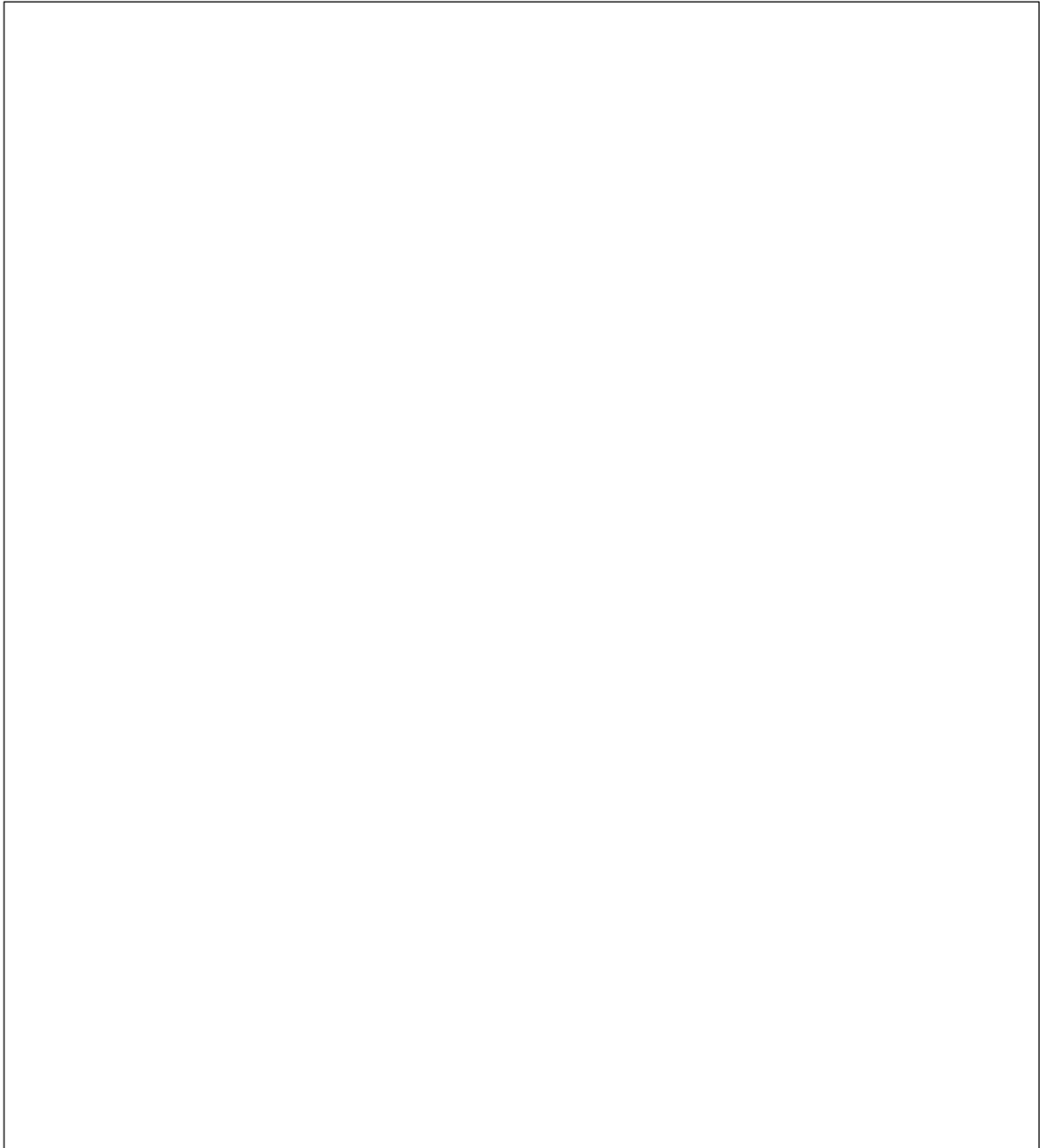


Image Source: <https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html>

Algorithm

- Import dataset
- Create the network model architecture
- Build the layers and compile the model
- Fit the values and train with set number of epochs and in batch.
- Predict the outcome using test data
- Check the performance metrics of the model

Code Implementation:**Result:**

Therefore, the CNN model with its individual layers has been successfully created and tested for image recognition. This was achieved by training the model using a dataset and obtaining performance metrics for the model.

Experiment No: 9 Stock Price Prediction Using LSTM

Aim:

Construct the LSTM model using layers to predict stock prices.

Description:

LSTM Model

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture designed to capture long-term dependencies in sequential data. It is particularly effective for tasks such as natural language processing (NLP), speech recognition, and time series forecasting. Here's how LSTM works and how it can be implemented in TensorFlow 2:

Cell State and Gates:

- The key to LSTM's ability to capture long-term dependencies is its cell state, which serves as a conveyor belt that can run along the entire sequence, with only minor linear interactions.
- LSTM has three gates that control the flow of information into and out of the cell state:
 - Forget Gate: Determines which information to discard from the cell state.
 - Input Gate: Determines which new information to store in the cell state.
 - Output Gate: Determines which information to output based on the cell state.

LSTM Operations:

- At each time step, the LSTM takes as input the current input x_t , the previous hidden state $h(t-1)$, and the previous cell state $c(t-1)$.
- It then computes the new cell state c_t and the new hidden state h_t using the following equations:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t$$

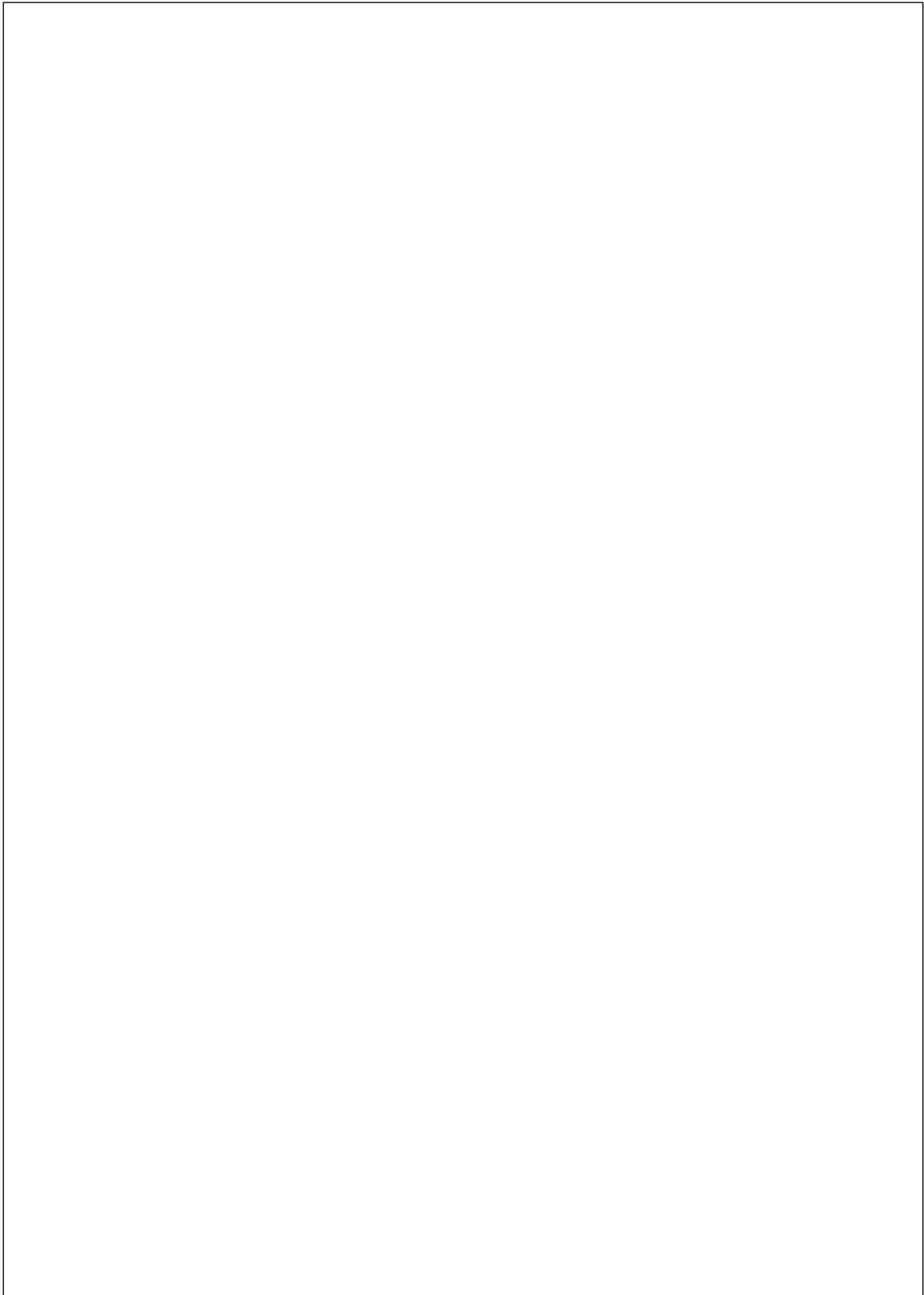
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \cdot \tanh(c_t)$$

Here, f_t , i_t , and o_t are the forget, input, and output gate vectors, respectively. \tilde{c}_t is the candidate cell state vector, and c_t is the new cell state. σ is the sigmoid activation function, and \tanh is the hyperbolic tangent activation function.

Algorithm

- Import dataset
- Feature scale the data if needed
- Create the network model architecture
- Build the layers and compile the model
- Fit the values and train with set number of epochs and in batch.
- Predict the outcome using test data

Code Implementation:**RESULT:**

Thus, the LSTM model generated predicts the stock price based on the provided data.

Experiment No: 10 Neural Style Transfer Using Gan

Aim:

Applying a style from one image to another using neural style transfer with Generative Adversarial Networks (GANs).

Description:

Neural Style Transfer:

Neural style transfer is a technique in deep learning that combines the content of one image with the style of another image to create a new image. This technique is based on the idea that the content of an image can be separated from its style, and these two aspects can be independently manipulated.

Content Image and Style Image:

- Neural style transfer requires two input images: a content image and a style image.
- The content image is the image whose content (objects and their arrangement) we want to preserve in the final image.
- The style image is the image whose style (textures, colors, and brush strokes) we want to apply to the final image.

Feature Extraction:

- Neural style transfer typically uses a pre-trained convolutional neural network (CNN) to extract features from the content and style images.
- The CNN is used to extract features at different layers of the network, capturing both low-level features (e.g., edges and textures) and high-level features (e.g., objects and their arrangements).

Loss Functions:

- Neural style transfer uses three loss functions to define the optimization problem:
- **Content Loss:** Measures the difference between the content features of the generated image and the content image. It ensures that the generated image preserves the content of the content image.
- **Style Loss:** Measures the difference between the style features of the generated image and the style image. It ensures that the generated image has a similar style to the style image.
- **Total Variation Loss:** Encourages smoothness in the generated image by penalizing rapid changes in pixel values.

Optimization:

- Neural style transfer aims to minimize the total loss, which is a weighted sum of the content loss, style loss, and total variation loss.
- The optimization process involves updating the pixel values of the generated image to minimize the total loss using gradient descent or a similar optimization algorithm.

Output Image:

- The output of the neural style transfer algorithm is a new image that combines the content of the content image with the style of the style image.
- The generated image preserves the content and structure of the content image while incorporating the colors, textures, and brush strokes of the style image.

Neural style transfer is a powerful technique that can be used for artistic purposes, such as creating stylized images or videos, as well as for practical applications, such as image editing and content generation.



Image Source - <https://chelseatroty.com/2019/03/01/neural-style-transfer-with-latte-art-part-1-layer-depth/>

Algorithm

- Import original image
- Check the size, shape, color channel and tensor shape
- Import Style image and perform the same set of checks
- Display and check the actual images
- For the model connect to the TF-Hub handle
- To the model now provide the original and style image
- Finally display the styled image

Code Implementation:



RESULT:

Therefore, the styled image result is achieved through the utilization of neural style transfer with GANs.