Poznań University of Technology

Faculty of Automatic Control, Robotics and Electrical Engineering

Smart Aerospace and Autonomous Systems

BASICS OF SMART SYSTEMS

Inz. Safarini Mohammed

Report of Laboratory Task 5

Mini Tracker Robot following the light spot position

Students:

159010 - JIOKENG I TOKO Jean Junior Maldini

162943 - FIKRASELASSIE Eshetu Seid

Contents

I. Introduction and motivations

The objective of this project is to develop a neural network-based control system for a mobile robot that can track and follow a light source. The robot is equipped with three sensors, each capable of detecting the light source within a specific range.The neural network will be trained to process the input data from these sensors and generate appropriate control signals to navigate the robot towards the light source. By leveraging the computational power and learning capabilities of neural networks, the robot should be able to adapt to varying light conditions and positions, enabling it to effectively pursue and maintain proximity to the light source.

II. Implementation of the Model

1. The Neural Network Training Model

a. Setup of Data

We computed first, **the Input vector P** created using sensor_model() function, simulating readings from robot's three sensors. Input vector represents sensor data network will process.

```
% Initialize the input data matrix 'P' as a column vector with 3 zeros
P = [];
```

Secondly, we defined **the Target vector T** created using velocity_estimator() function, calculating appropriate velocities for left and right wheels based on desired trajectory and light source position. Target vector represents desired output.

```
% Initialize the target data matrix 'T' as a column vector with [-1; 1]
T = [];
```

Now, we will fill these two vectors according to the random values distance and angle of the light.

```
    global netN

    % Generate random angles and distances
    alpha = randi([-54 54], 1, 100);
    distance_values = randi([5 55], 1, 100);

    % Initialize input and target datasets
    P = [];
    T = [];

    Range = [0 5;0 5;0 5];

    % Prepare input set and target set
    for i = 1:100
        angle = alpha(i);
        distance = distance_values(i);

        measurement = sensor_model([angle, distance]);
        velocities = velocity_estimator([angle, distance]);

        P = [P, measurement];
        T = [T, velocities];
    end
```

With input and target vectors prepared, we write train_net() code to create and train our neural network. During training, sensor values fed as input, corresponding target wheel velocities used as expected output. Through supervised learning, neural network learned to map sensor inputs to desired wheel velocities, enabling robot to follow light source.

From the function Sensor_model(), we obtain a vector of 3 coordinates, each value is included into 0 and 5. Thus will help for defining the range of the input data.

For our learning process, we will sets the activation functions for the hidden and output layers, respectively **Logsig** which takes a matrix of net input vectors, P and returns one matrix A and the elements of P squashed into [0, 1] and **Tansig** defined by **tansig(x)= 2/(1+exp(-2*x))-1**.

```
%% Neural Network Section For Robot
global netN

% Create a new feed-forward neural network using the 'newff' function
netN = newff([0 5; 0 5; 0 5], [21 2], {'logsig', 'tansig'});
```

b. Velocities of the wheels of our Robot

To drive our robot, we will change the velocities of the wheels. The `velocity_estimator` function will determine the velocities of the left and right wheels of a mobile robot based on the angle and distance at which a light source is detected. The function takes the angle and distance as inputs.

```
function velocity = velocity_estimator(in)

% compute right and left velocities
if (angle > -3) && (angle < 3) % If the angle is between -3 and 3 degrees
% The robot should move forward
    vr = 1; % Set the right velocity to 1
    vl = 1; % Set the left velocity to 1

elseif (angle >= 3) && (angle <= 54) % If the angle is between 3 and 54 degrees
% The robot should move in the left side
    vr = 1; % Set the right velocity to 1
    vl = 0; % Set the left velocity to 0

elseif (angle <= -3) && (angle >= -54)
    % If the angle is between -3 and -54 degrees
% The robot should move in the right side

```

```
    vr = 0; % Set the right velocity to 0
    vl = 1; % Set the left velocity to 1
end

velocity = [vr;vl] ; % Create a column vector with the computed velocities

end
```

c. Training our created Model

Launch the training of our model and save it in the one variable

```
…
    % Initialize and train the neural network
    netN = newff([0 5; 0 5; 0 5], [21 2], {'logsig', 'tansig'});
    netN = train(netN, p, T);
```

2. Prediction of the Robot

a. Relative Location

The `relative_location` function will calculate the relative angle and distance between a light source and a robot. It takes five inputs: the x and y coordinates of the light source (`xl` and `yl`), the x and y coordinates of the robot (x and y), and the robot's orientation or heading angle (`th`) in radians.

- The distance between the light source and the robot using **the Pythagorean Theorem**.
- The angle between the light source and the robot's heading using the `atan2` function. This angle is initially in radians, so we will convert it to degrees.

Finally, the function returns a row vector containing the relative angle (in degrees) and the distance between the light source and the robot.

```
Function out = relative_location(in)

    xl = in(1); % X-coordinate of the light source
    yl = in(2); % Y-coordinate of the light source
    x = in(3); % X-coordinate of the robot
    y = in(4); % Y-coordinate of the robot
    th = in(5); % Orientation (heading angle) of the robot in radians

     % Calculate the distance between the light source and the robot
     d = sqrt((xl - x)^2 + (yl - y)^2);
     % Calculate the angle between the light source and the robot's heading (in radians)
     q = atan2((yl - y), (xl - x));
    % The angle is in radians, but we need to convert it to degrees
    a_rd = q -th; % Relative angle in radians
    a_deg = (a_rd*180.0) / pi; % Convert relative angle from radians to degrees

    a = a_deg;
    out = [a d]; % Return the relative angle and distance as a row vector
end
```

b. Prediction of our controller

Our controller is based on the neural network that we created. After training we will use the result it to make prediction, calculation and control of the velocities of our robot :

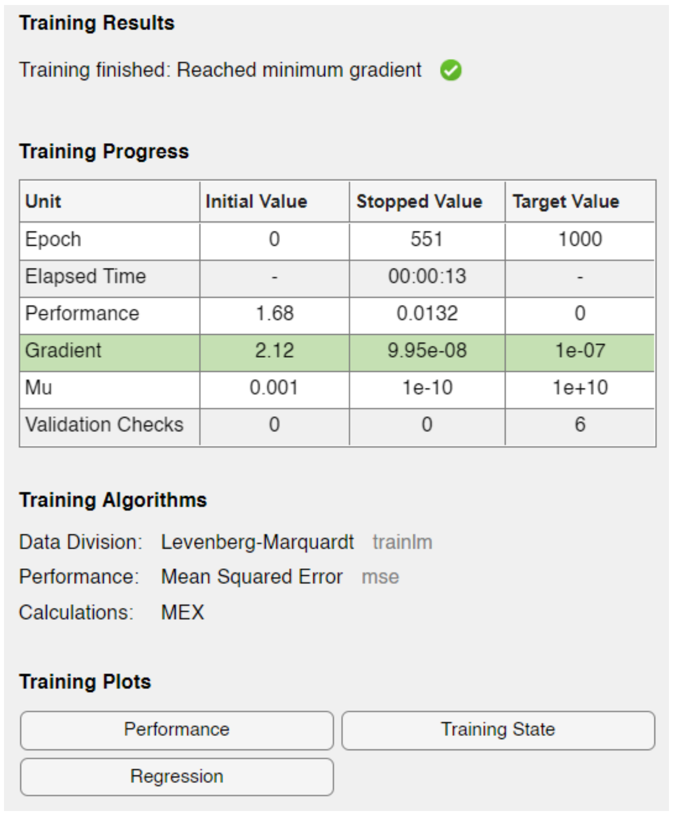
```
% Find the estimated velocities by ANN
function velocities = control_net(sensor1, sensor2, sensor3)

global netN

p=[sensor1; sensor2; sensor3];

velocities=sim(netN,p); % Return the velocities of each robot wheel to reach the light
```

c. Results of our trained Model



Training Results

Training finished: Reached minimum gradient ✓

Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	551	1000
Elapsed Time	-	00:00:13	-
Performance	1.68	0.0132	0
Gradient	2.12	9.95e-08	1e-07
Mu	0.001	1e-10	1e+10
Validation Checks	0	0	6

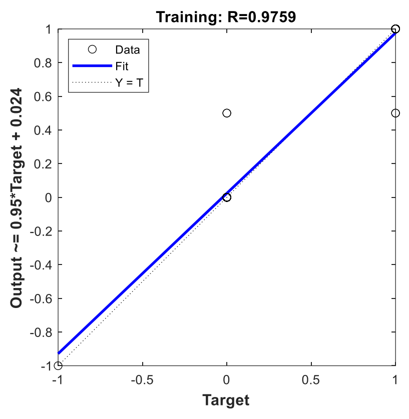
Training Algorithms

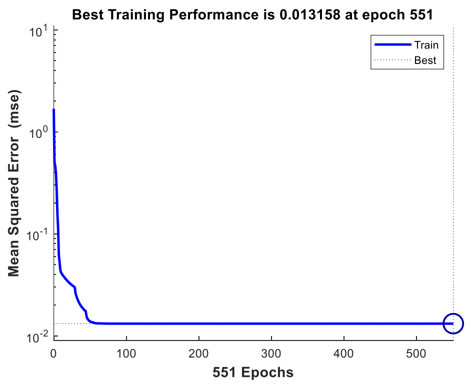
Data Division: Levenberg-Marquardt trainlm

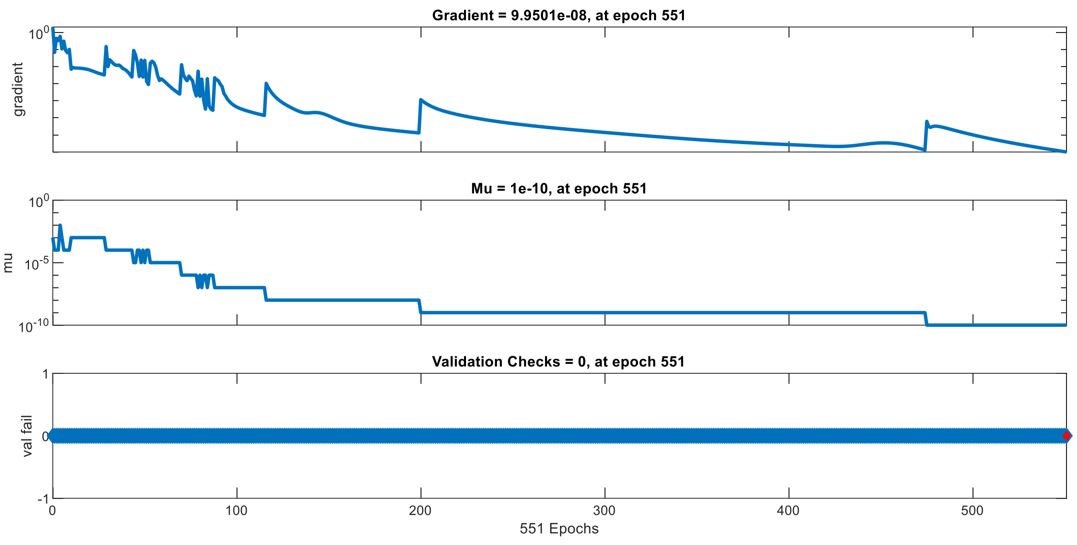
Performance: Mean Squared Error mse

Calculations: MEX

Training Plots





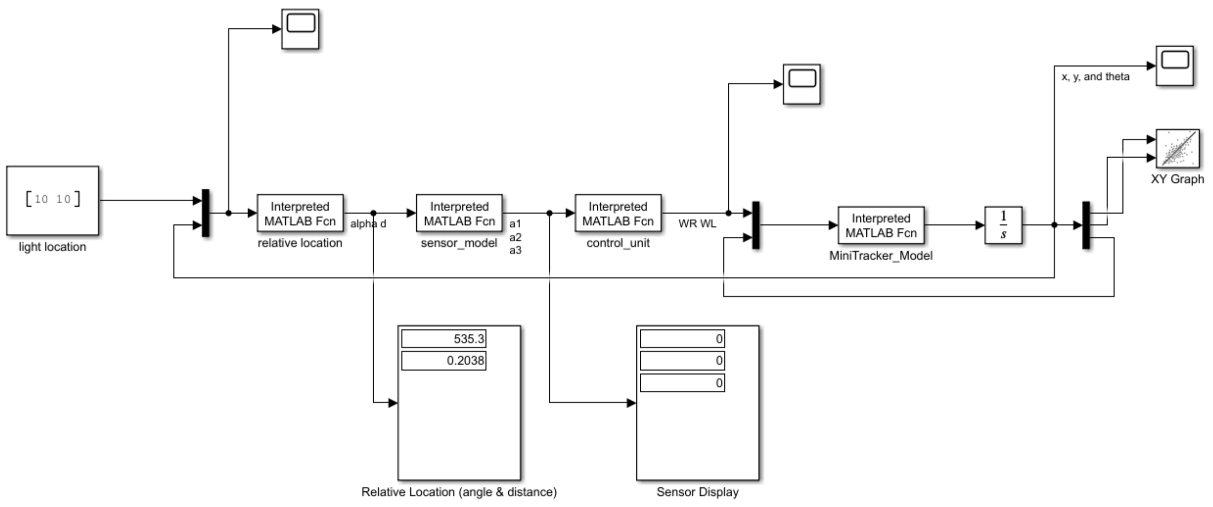


Our model very well trained the data we computed.

III. Results and comments

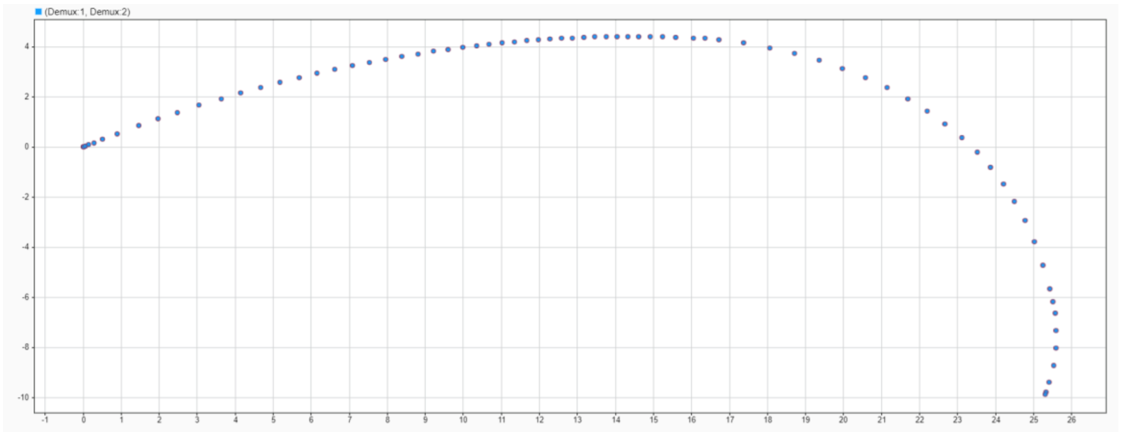
1. Simulation scheme

This whole system is simulated in Simulink and MATLAB.

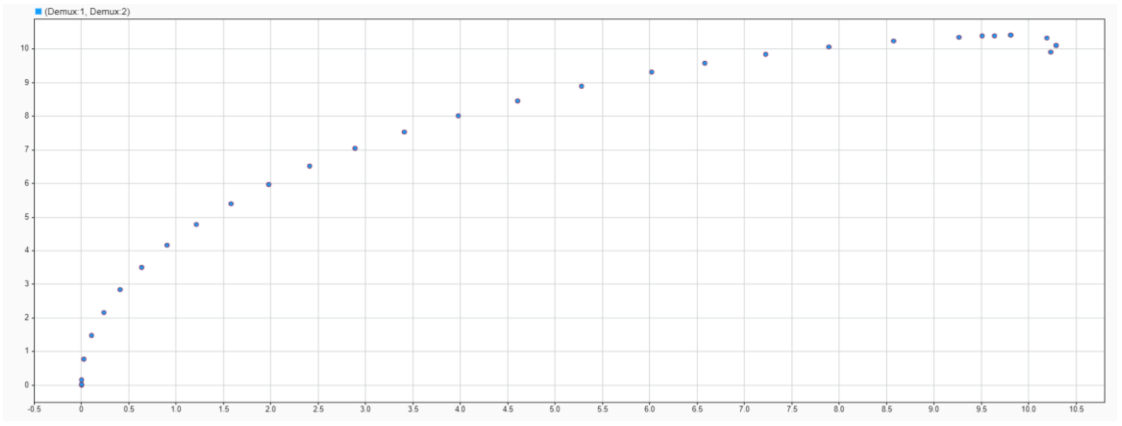


2. Tests of our Model

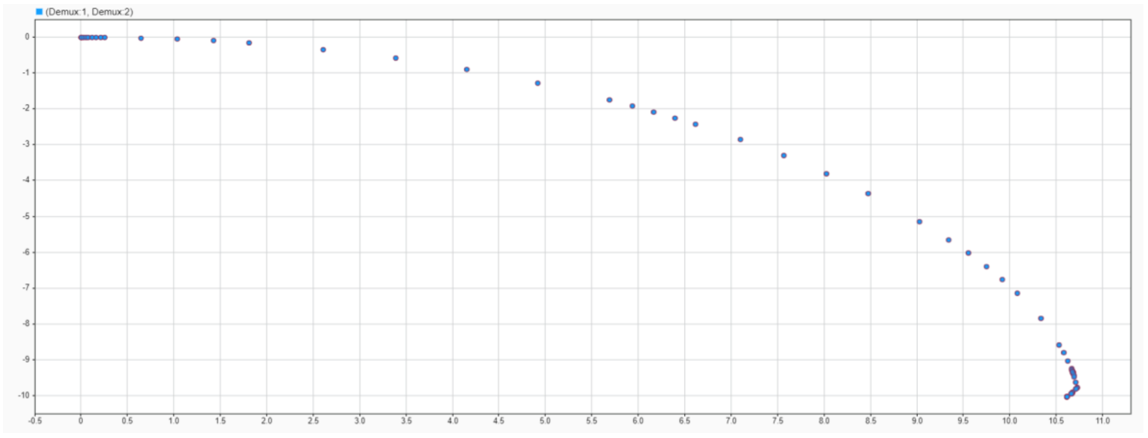
a. First Position of the Light: [25 -10]



b. Second Position of the Light: [10 10]



c. Third Position of the light: [-10 10]



3. Comments

Our model reaches the position of the light. Not exactly, but it is very closed to the true value we computed. The reason can be explained by 2 ways:

- **Lack of training data:** To enhance the performance of our controller. We took only 36 values; it can be less for perfect performance.
- **Dynamics of the robot:** Mostly in the real world application, because of space of mobility or dumpling on wheels.

Conclusion

This report examined a neural network-based model for controlling the velocity of a two-wheeled robot to reach a light position. The results were very good, though not perfectly accurate.

Performance: The model effectively guided the robot towards the light with high accuracy.

Adaptability: The neural network can adapt into varying conditions and disturbances.

Precision: The robot occasionally missed the exact target position, indicating room for improvement. But it is not so bad because the light can cover a surface.

The neural network-based control model shows strong potential for robotic navigation, demonstrating adaptability and robust performance. While it achieved very good results, future work on refining the model could enhance precision. Overall, this approach is promising for complex control tasks in robotics.