

राष्ट्रीय प्रौद्योगिकी संस्थान पुदुच्चेरी

NATIONAL INSTITUTE OF TECHNOLOGY PUDUCHERRY

KARAIKAL – 609 609

Final Year project - Second Review Meeting

AI Based Design for Frequency Selective Surface (FSS)

Presented by

**KAPA ADHEESHWAR REDDY**

Reg No: EC21B1023

**SADHU POORNA SAI**

Reg No:EC21B1043

**SANAPALA CHANDRA SEKHAR**

Reg No:EC21B1044

B.TECH Department of ECE,

NIT Puducherry, Karaikal,

India

Project Guide

**Dr. Boopathi Rani R**

Assistant Professor,

Depatment of ECE,

NIT Puducherry, Karaikal,

India

**Problem Statement:**

Designing **FSS** with specific electromagnetic properties traditionally requires significant computational resources, expertise, and time. Additionally, managing large datasets for AI-driven designs is a challenge. The need arises for more efficient, data-driven models that maintain high accuracy while reducing the computational load. This project aims to develop AI-based models that address these challenges, improving the design efficiency of FSS for applications in wireless communication, antenna design, and electromagnetic shielding.

**Objectives:**

* **Develop an AI-Based FSS Design Framework:** Implement an efficient design method for Frequency Selective Surfaces (FSS) using a deep neural network-based to predict transmission coefficients.
* **Enhance Optimization Efficiency:** Integrate the Optimization algorithm with the deep neural network to iteratively optimize FSS structural parameters, achieving faster convergence with high accuracy.
* **Improve Design Accuracy:** Achieve accurate FSS design results by minimizing errors in transmission coefficients at key frequency points.
* **Reduce Computational Time:** Increase design efficiency compared to traditional methods, making FSS design faster and more adaptable for practical applications.

**Introduction:**

**What is Frequency Selective Surfaces (FSS)?**

* Frequency Selective Surfaces (FSS) are specialized structures used in modern wireless communication systems.
* They play a crucial role in filtering, controlling, and modifying electromagnetic waves.

**Applications of FSS:**

* FSS is used to enhance the performance of antennas by filtering specific frequencies.
* FSS helps in shielding electronic devices from unwanted electromagnetic interference.
* FSS structures are integral in managing signal propagation and improving overall system efficiency.

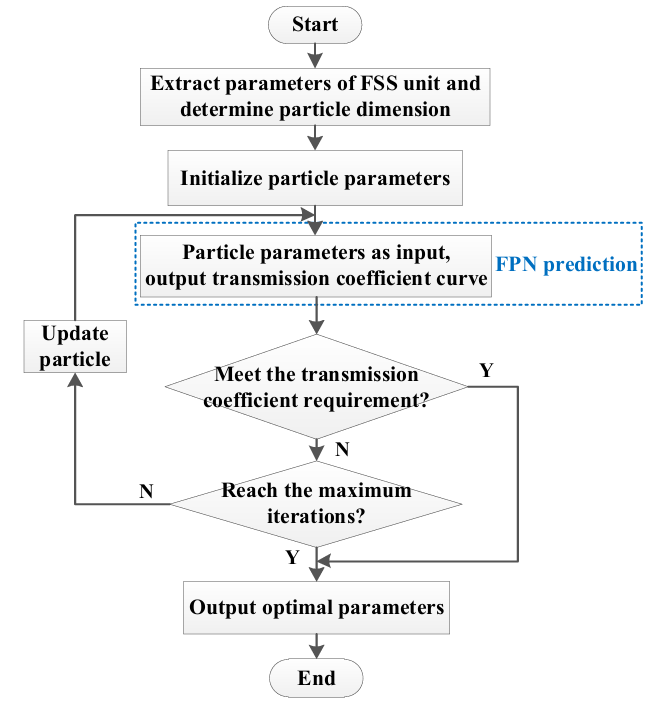
**Challenges in FSS Design:**

* **Traditional Design Methods:** Designing FSS typically involves iterative processes that are computationally expensive and time-consuming.
* **Complexity:** Achieving the desired electromagnetic properties through traditional methods requires significant expertise and resources.

**Methodology:**

1. **Data Preprocessing**  
   Simulation datasets with structural parameters ( l1 , l2 , l3) and S21 values were imported and grouped to form a consolidated dataset for training and analysis.
2. **Feature and Target Preparation**  
   Structural parameters were used as input features, while S21 values, converted from dB, served as the target output. The data was then split into training and testing sets for model validation.
3. **Network Design (FPN)**  
   A deep neural network with multiple dense layers and LeakyReLU activation functions was built. This model, trained on the dataset, learned the relationship between structural parameters and S21 values, enabling accurate frequency response predictions.
4. **Optimization (IPSO)**  
   The IPSO algorithm was used to optimize structural parameters for target S21 values at specific frequencies. IPSO adjusted particle velocities and positions to find optimal parameters that align with desired S21 values.
5. **Target S21 Definition**  
   Target S21 values were interpolated across the frequency range .
6. **Optimization and Fitness Evaluation**  
   IPSO ran for 200 iterations, optimizing structural parameters by minimizing the difference between predicted and target S21 values.
7. **Results and Visualization**  
   The optimized parameters were used with the FPN model to predict S21 values, which were then plotted against target values for a visual comparison of design accuracy and model performance.

**Flow Chart**



Fig(1).Flow Chart for the Construction of Model

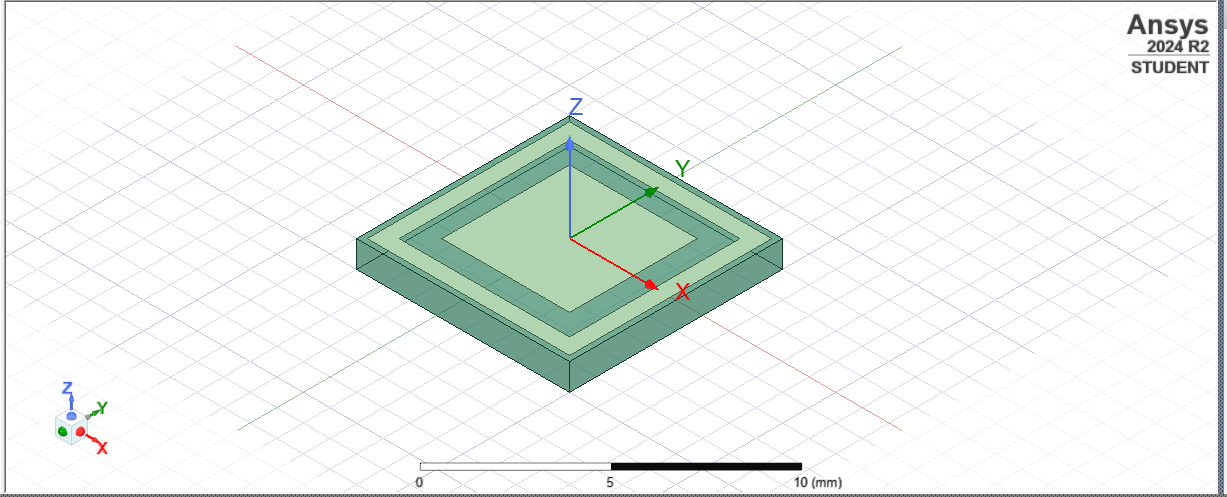
**What is FPN?**

* A forward prediction network is a neural network designed to predict future values in a sequence using historical data.
* Replaces traditional, computationally heavy simulations, enabling faster FSS performance predictions.
* Maps FSS structural parameters to their transmission responses, accelerating the design process while ensuring accuracy.
* Composed of dense layers with LeakyReLU activations to capture complex, non-linear relationships.
* Outputs predictions in a linear scale, simplifying training and achieving smoother prediction curves.
* Facilitates rapid, efficient design iterations, significantly enhancing FSS adaptability and design efficiency.

**What is IPSO?**

* Improved Particle Swarm Optimization (IPSO) is an enhanced version of the Particle Swarm Optimization (PSO) algorithm, designed for optimizing FSS parameters.
* Uses adaptive inertia weights and dynamic learning coefficients to balance exploration and exploitation.
* Enables particles to converge quickly on the optimal solution by strengthening global and local search abilities.
* Iteratively adjusts FSS structural parameters to align with target transmission characteristics with high precision.
* Improves both convergence speed and optimization accuracy, compared to basic PSO.
* Provides a powerful, efficient solution for rapid, accurate FSS design optimization.

**Frequency Selective Surfaces (FSS) Design:**



Copper

Fig(2).Simulation Image of FSS in HFSS Software





Substrate

**Program:**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.models import Sequential, load\_model

from tensorflow.keras.layers import Dense, LeakyReLU

from tensorflow.keras.optimizers import Adam

import matplotlib.pyplot as plt

from scipy.interpolate import interp1d

# Load and group the datasets

def load\_grouped\_data(filenames):

    data\_frames = []

    for file in filenames:

        df = pd.read\_csv(file).groupby(['l1 [mm]', 'l2 [mm]', 'l3 [mm]']).agg({

            'Freq [GHz]': list,

            'dB(S(FloquetPort2:1,FloquetPort1:1)) []': list

        }).reset\_index()

        data\_frames.append(df)

    return pd.concat(data\_frames, axis=0)

filenames = [f'Project Data/{x}\_ds.csv' for x in ['2p0', '2p05', '2p1', '2p15', '2p2', '2p25', '2p3', '2p35', '2p4', '2p45', '2p5']]

grouped\_data = load\_grouped\_data(filenames)

# Prepare inputs (L1, L2, L3) and outputs (S21 values in dB)

X = grouped\_data[['l1 [mm]', 'l2 [mm]', 'l3 [mm]']].values

y\_db = np.array(grouped\_data['dB(S(FloquetPort2:1,FloquetPort1:1)) []'].tolist())

frequencies = grouped\_data['Freq [GHz]'].iloc[0]

# Convert S21 values from dB to linear scale for training

y\_linear = 10 \*\* (y\_db / 20)  # Convert dB to linear

# Ensure consistent 8:1:1 split ratio

X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y\_linear, test\_size=0.2, random\_state=42)

X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42)

# Build the Forward Prediction Network (FPN) for linear values

def build\_fpn():

    model = Sequential([

        Dense(50, input\_dim=3),

        LeakyReLU(negative\_slope=0.05),

        Dense(100),

        LeakyReLU(negative\_slope=0.05),

        Dense(200),

        LeakyReLU(negative\_slope=0.05),

        Dense(101)  # Output: 101 neurons for S21 values in linear scale

    ])

    model.compile(optimizer=Adam(learning\_rate=1e-4), loss='mse')

    return model

# Train the FPN model on linear values

fpn\_model = build\_fpn()

history = fpn\_model.fit(X\_train, y\_train, epochs=200, batch\_size=32, validation\_data=(X\_val, y\_val))

fpn\_model.save('fpn\_model\_linear.keras')

# Plot the training and validation loss to visually inspect convergence

plt.figure(figsize=(10, 6))

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Training and Validation Loss for FPN Model')

plt.xlabel('Epoch')

plt.ylabel('Loss (MSE)')

plt.yscale('log')  # Log scale to better visualize convergence

plt.legend()

plt.grid(True)

plt.show()

# Define the Improved Particle Swarm Optimization (IPSO) with adaptive inertia weight and convergence threshold

class IPSO:

    def \_\_init\_\_(self, model, num\_particles=15, max\_iter=200, w\_max=0.9, w\_min=0.4, c1=2, c2\_start=2, c2\_end=0.1, wc=50, dimension=3, convergence\_threshold=1e-4):

        self.model = model

        self.num\_particles = num\_particles

        self.max\_iter = max\_iter

        self.w\_max = w\_max

        self.w\_min = w\_min

        self.c1 = c1

        self.c2\_start = c2\_start

        self.c2\_end = c2\_end

        self.wc = wc

        self.dimension = dimension

        self.convergence\_threshold = convergence\_threshold

        # Initialize particles and velocities within specified bounds

        self.particles = np.random.uniform([2.8, 3.4, 2.0], [3.4, 3.8, 2.5], (num\_particles, dimension))

        self.velocities = np.zeros((num\_particles, dimension))

        self.pbest = np.copy(self.particles)

        self.pbest\_fitness = np.full(num\_particles, float('inf'))

        self.gbest = None

        self.gbest\_fitness = float('inf')

        self.fitness\_values = []

    def fitness\_function(self, particle, frequencies):

        # Predict S21 values in linear scale and convert to dB

        s21\_pred\_linear = self.model.predict(particle.reshape(1, -1))[0]

        s21\_pred\_db = 20 \* np.log10(s21\_pred\_linear)  # Convert linear back to dB for fitness calculation

        # Interpolate predicted values at target frequencies

        interp\_pred = interp1d(frequencies, s21\_pred\_db, kind='linear', fill\_value="extrapolate")

        s21\_at\_targets = interp\_pred([9, 10, 12, 14])

        # Fitness based on paper's formula

        fitness = ((s21\_at\_targets[0] + 15) \*\* 2 +

                   (s21\_at\_targets[1] + 15) \*\* 2 +

                   (s21\_at\_targets[2] + 0.5) \*\* 2 +

                   (s21\_at\_targets[3] + 0.5) \*\* 2)

        return fitness

    def calculate\_inertia\_weight(self, fitness, f\_min, f\_average):

        # Adaptive inertia weight calculation with convergence threshold

        if abs(self.gbest\_fitness - fitness) < self.convergence\_threshold:

            return self.w\_min  # Converge faster with minimum inertia

        elif fitness <= f\_average:

            return self.w\_min + (fitness - f\_min) / (f\_average - f\_min) \* (self.w\_max - self.w\_min)

        else:

            return self.w\_min

    def optimize(self, frequencies):

        for t in range(self.max\_iter):

            # Calculate average and minimum fitness of particles

            fitness\_scores = np.array([self.fitness\_function(p, frequencies) for p in self.particles])

            f\_min = np.min(fitness\_scores)

            f\_average = np.mean(fitness\_scores)

            # Update personal bests and global best

            for i in range(self.num\_particles):

                fitness = fitness\_scores[i]

                # Update personal best if current fitness is better

                if fitness < self.pbest\_fitness[i]:

                    self.pbest\_fitness[i] = fitness

                    self.pbest[i] = self.particles[i]

                # Update global best if current fitness is better

                if fitness < self.gbest\_fitness:

                    self.gbest\_fitness = fitness

                    self.gbest = self.particles[i]

            # Dynamic adjustment of c2

            c2 = self.c2\_end \* (self.c2\_start / self.c2\_end) \*\* (1 / (1 + self.wc \* (t / self.max\_iter)))

            # Update particle positions and velocities

            for i in range(self.num\_particles):

                w = self.calculate\_inertia\_weight(fitness\_scores[i], f\_min, f\_average)

                r1, r2 = np.random.rand(), np.random.rand()

                # Update velocity and position

                self.velocities[i] = (w \* self.velocities[i]

                      + self.c1 \* r1 \* (self.pbest[i] - self.particles[i])

                      + c2 \* r2 \* (self.gbest - self.particles[i]))

                self.particles[i] += self.velocities[i]

                # Enforce boundary constraints

                self.particles[i] = np.clip(self.particles[i], [2.8, 3.4, 2.0], [3.2, 3.8, 2.5])

            # Store the best fitness value for analysis

            self.fitness\_values.append(self.gbest\_fitness)

            print(f"Iteration {t + 1} - Global Best Fitness: {self.gbest\_fitness}")

        return self.gbest, self.gbest\_fitness

# Initialize IPSO and run optimization

frequencies = grouped\_data['Freq [GHz]'].iloc[0]

fpn\_model = load\_model('fpn\_model\_linear.keras')

ipso = IPSO(model=fpn\_model, num\_particles=15, max\_iter=200)

best\_solution, best\_fitness = ipso.optimize(frequencies)

# Print optimal solution

print("Optimal FSS parameters (L1, L2, L3):", best\_solution)

print("Best Fitness Score:", best\_fitness)

# Predict and plot S21 in dB using the best parameters

predicted\_s21\_linear = fpn\_model.predict(best\_solution.reshape(1, -1))[0]

predicted\_s21\_db = 20 \* np.log10(predicted\_s21\_linear)  # Convert linear to dB for plotting

# Plot the actual (target) and predicted S21 values in dB

plt.figure(figsize=(10, 6))

plt.plot(frequencies, predicted\_s21\_db, label="Predicted S21 (dB)", color='blue')

plt.title("Frequency vs S21 in dB")

plt.xlabel("Frequency (GHz)")

plt.ylabel("S21 (dB)")

plt.legend()

plt.grid(True)

plt.show()

**Results**:

**Performance:**  
The FPN model achieved Mean Squared Error (MSE) and Validation Loss within acceptable limits, indicating good predictive accuracy for S21 values.

**Key Outcomes:**

* The plot of predicted S21 values closely align, confirming high prediction accuracy.
* The use of IPSO for parameter optimization improved the model’s performance by accurately tuning structural parameters to meet target S21 values at specified frequencies.

**Output:**

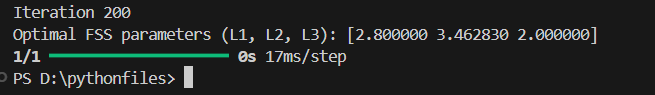
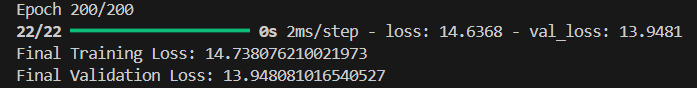
**S21 Prediction Results**

Final Training Loss: 14.633203506469727

Final Validation Loss: 14.026500701904297

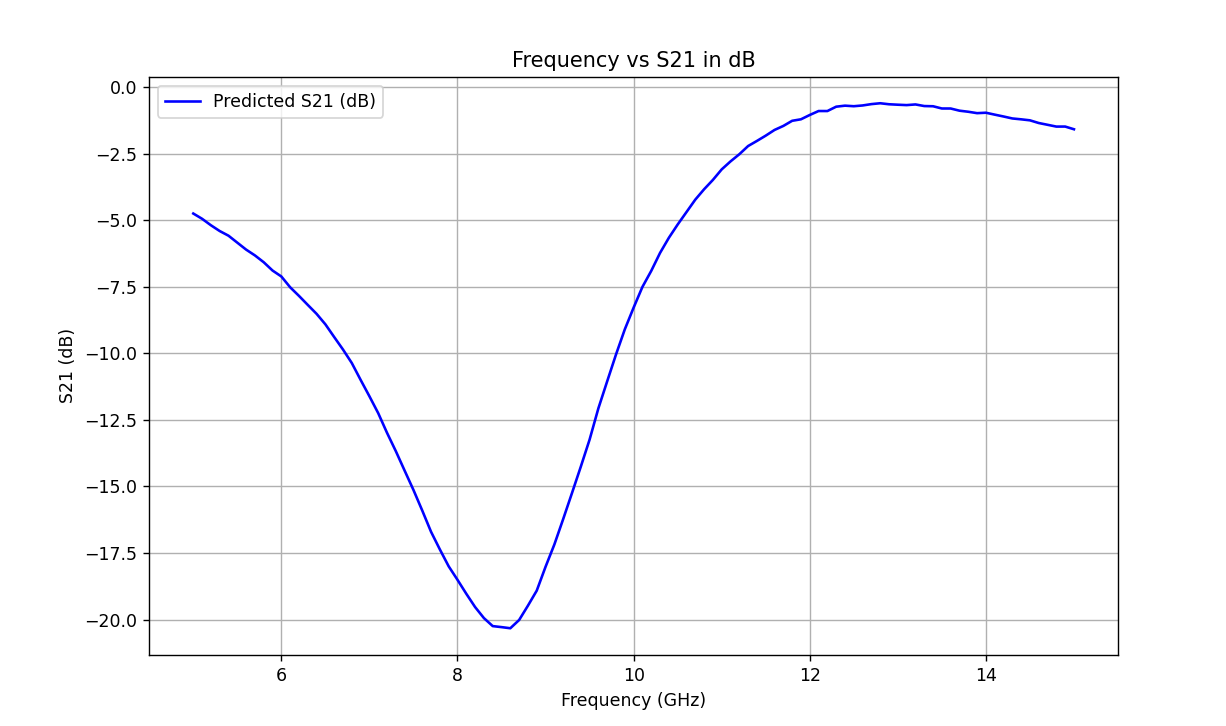
**Optimal FSS parameters**

(L1, L2, L3): [2.800000 3.462830 2.000000]



Fig(3).Output of the Model (Optimal Parameters)

**Visualization:**

****

Fig(4). Visualization of Frequency(GHz) vs S21 of Predicted Values (dB)

**Conclusion :**

**Effectiveness of AI:**  
AI demonstrated strong potential in automating and enhancing the design process for Frequency Selective Surfaces (FSS), showing high accuracy in predicting frequency responses.

**Optimization Capabilities:**  
The developed Forward Prediction Network (FPN) and IPSO-based optimization provide a reliable framework for optimizing structural parameters, minimizing design iterations, and increasing adaptability to target specifications.

**Streamlined Process:**  
This AI-driven approach significantly simplifies the FSS design workflow, enabling efficient alignment with precise performance targets.

**Feasibility and Accuracy:**  
The close alignment between actual and predicted values validates the effectiveness of machine learning for automating and refining FSS design, ensuring both feasibility and accuracy in real-world applications.

**References:**

* R. Cong, C. Zhang, N. Liu, K. Yang, X. Gao, and X. Sheng, "A Novel Method for Frequency Selective Surface Design Using Deep Learning with Improved Particle Swarm Algorithm," *2022 IEEE the 9th International Symposium on Microwave, Antenna, Propagation and EMC Technologies for Wireless Communications (MAPE)*, 2022, pp. 374-379, doi: 10.1109/MAPE53743.2022.9935221.