NeRF-based Intraoperative Registration

Bachelor's Thesis

Your Name

Department of Computer Science
Harvard University

Supervisor: Professor Name

March 15, 2025

Abstract

This thesis explores the application of Neural Radiance Fields (NeRFs) for intraoperative brain registration. Unlike traditional mesh-based approaches, NeRFs provide a differentiable, implicit function of the brain's geometry and appearance, enabling optimization of camera positions through backpropagation for precise alignment of preoperative and intraoperative brain images. We investigate various loss functions and hypernetwork style transfer techniques to improve registration accuracy.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Intraoperative brain registration is a critical component of image-guided neurosurgery, enabling surgeons to navigate with precision by aligning preoperative imaging data with the current state of the patient's brain during surgery. Traditional registration methods often rely on rigid or non-rigid transformations of mesh-based models, which can be computationally expensive and lack the flexibility needed to account for brain shift and deformation that occurs during surgery [2].

This thesis explores a novel approach to intraoperative registration using Neural Radiance Fields (NeRFs) [3]. NeRFs provide an implicit, differentiable representation of 3D scenes, allowing for efficient optimization of camera positions through backpropagation. By leveraging the differentiable nature of NeRFs, we can iteratively refine camera poses to align rendered views with intraoperative images, potentially offering more accurate and efficient registration compared to traditional methods.

1.2 Problem Statement

The core challenge addressed in this work is how to accurately align preoperative brain models with intraoperative images in real-time, accounting for brain shift and deformation. Specifically, we investigate:

- How can NeRFs be effectively utilized for intraoperative brain registration?
- What loss functions provide the most accurate and robust registration results?

- How can hypernetwork-based style transfer improve cross-modal registration between different imaging modalities?
- What are the computational requirements and limitations of NeRF-based registration in a clinical setting?

1.3 Contributions

This thesis makes the following contributions to the field of medical image registration:

1. Implementation of a NeRF-based intraoperative registration framework that is agnostic to the specific NeRF implementation
2. Comprehensive evaluation of various loss functions for registration accuracy
3. Novel application of hypernetwork-based style transfer for cross-modal registration
4. Analysis of computational efficiency and clinical feasibility of the proposed methods

1.4 Thesis Structure

The remainder of this thesis is organized as follows:

Chapter 2 provides background information on neural radiance fields, medical image registration, and related work in the field.

Chapter 3 details the methodology of our approach, including the mathematical formulation of the registration problem and the proposed solutions.

Chapter 4 describes the implementation details of our framework, built on top of nerfstudio.

Chapter 5 presents the experimental setup and evaluation metrics used to assess the performance of our methods.

Chapter 6 reports the results of our experiments and provides a comparative analysis of different loss functions and hypernetwork approaches.

Chapter 7 discusses the implications of our findings, limitations of the current approach, and potential directions for future work.

Chapter 8 concludes the thesis with a summary of our contributions and their significance to the field.

Chapter 2

Background and Related Work

2.1 Neural Radiance Fields

Neural Radiance Fields (NeRFs) [3] represent a paradigm shift in 3D scene representation. Unlike traditional explicit representations such as meshes or voxel grids, NeRFs encode scenes as continuous volumetric functions using neural networks. This section provides an overview of NeRFs and their applications.

2.1.1 NeRF Fundamentals

A Neural Radiance Field is a continuous 5D function that maps a 3D location (x, y, z) and viewing direction (θ, ϕ) to a color (r, g, b) and volume density σ. This function is typically implemented as a multi-layer perceptron (MLP):

$$F_\Theta : (x, y, z, \theta, \phi) \rightarrow (r, g, b, \sigma) \tag{2.1}$$

where Θ represents the parameters of the neural network. To render an image from a specific camera pose, rays are cast through each pixel of the virtual camera. Points are sampled along each ray, and the network predicts the color and density at each point. These predictions are then composited using volume rendering techniques to produce the final pixel color.

2.1.2 NeRF Variants and Improvements

Since the introduction of the original NeRF, numerous variants have been proposed to address its limitations:

- **Instant-NGP** [4]: Accelerates NeRF training and rendering using a multiresolution hash encoding, reducing training time from hours to minutes.

- **Nerfstudio** [5]: A modular framework that provides abstractions for developing and experimenting with different NeRF implementations.
- **NeRFmm** [6]: Extends NeRFs to handle multi-view, multi-exposure images, which is particularly relevant for medical imaging where different modalities may be used.

2.2 Intraoperative Registration

Intraoperative registration is the process of aligning preoperative imaging data (such as MRI or CT scans) with the current state of the patient during surgery. This alignment is crucial for image-guided surgery, allowing surgeons to navigate with precision and avoid critical structures.

2.2.1 Traditional Registration Methods

Traditional registration methods can be categorized into:

- **Rigid Registration**: Assumes that the transformation between the preoperative and intraoperative images can be described by a combination of rotation and translation.
- **Non-rigid Registration**: Accounts for deformations by allowing local transformations, often modeled using splines or physical models.
- **Feature-based Registration**: Identifies and matches corresponding features (landmarks, edges, etc.) between the preoperative and intraoperative images.
- **Intensity-based Registration**: Optimizes a similarity measure between the intensity patterns of the images.

2.2.2 Challenges in Neurosurgical Registration

Brain registration during neurosurgery presents unique challenges:

- **Brain Shift**: The brain can deform significantly after the skull is opened due to gravity, loss of cerebrospinal fluid, and tissue manipulation.

- **Time Constraints**: Registration must be performed quickly to be useful during surgery.
- **Limited Intraoperative Imaging**: High-quality imaging modalities may not be available in the operating room.
- **Cross-modal Registration**: Preoperative MRI must often be registered with intraoperative ultrasound or optical images.

2.3 NeRF-based Registration

Recent work has explored the use of NeRFs for registration tasks, leveraging their differentiable nature for optimization.

2.3.1 iNeRF

iNeRF [10] inverts the traditional NeRF process by optimizing camera poses to match observed images rather than optimizing the NeRF parameters to match known poses. This approach has been demonstrated for pose estimation in general scenes but has limitations in medical contexts.

2.3.2 Cross-Modal Inverse Neural Rendering

Wang et al. [7] proposed a method for intraoperative registration using cross-modal inverse neural rendering. Their approach addresses the challenge of registering preoperative MRI data with intraoperative optical images by using a neural renderer to bridge the modality gap.

2.3.3 Parallel Inversion

Parallel Inversion [8] improves upon iNeRF by performing multiple inversions in parallel, increasing robustness to local minima and improving convergence speed. This approach is particularly relevant for real-time applications such as intraoperative registration.

2.4 Style Transfer and Hypernetworks

Style transfer techniques can be used to address the cross-modal nature of medical image registration by transforming images from one modality to appear as if they were acquired using another modality.

2.4.1 Neural Style Transfer

Neural style transfer [1] uses convolutional neural networks to separate and recombine the content and style of images. This technique has been applied to medical imaging to bridge the gap between different modalities.

2.4.2 Hypernetworks for Style Transfer

Hypernetworks are networks that generate the weights for another network. In the context of style transfer, a hypernetwork can be trained to generate style-specific parameters for a rendering network, allowing for efficient adaptation to different imaging modalities.

2.5 Loss Functions for Registration

The choice of loss function is critical for registration accuracy. Various loss functions have been proposed for medical image registration:

- **L1/L2 Loss**: Simple pixel-wise differences, which may not capture structural similarities.
- **Structural Similarity Index (SSIM)** [9]: Measures the structural similarity between images, accounting for luminance, contrast, and structure.
- **Mutual Information**: A statistical measure that quantifies the mutual dependence between two variables, useful for cross-modal registration.
- **Normalized Cross-Correlation**: Measures the similarity between two signals, invariant to linear transformations.

This thesis builds upon these foundations to develop and evaluate a comprehensive framework for NeRF-based intraoperative registration, with a focus on loss function exploration and hypernetwork-based style transfer for improved cross-modal registration.

Chapter 3

Methodology

This chapter presents the theoretical foundation and methodological approach of our NeRF-based intraoperative registration framework. We begin by formulating the registration problem, then describe our approach to solving it using Neural Radiance Fields, various loss functions, and hypernetwork-based style transfer.

3.1 Problem Formulation

The intraoperative registration problem can be formulated as finding the optimal transformation T that aligns a preoperative model with an intraoperative image. In our approach, we represent the preoperative model as a Neural Radiance Field F_Θ and seek to find the camera pose P that, when used to render an image from the NeRF, produces an image that best matches the intraoperative image I_{target}.

Mathematically, we can express this as:

$$P^* = \arg\min_P \mathcal{L}(R(F_\Theta, P), I_{target}) \tag{3.1}$$

where $R(F_\Theta, P)$ is the rendering function that produces an image from the NeRF F_Θ using camera pose P, and $\mathcal{L}$ is a loss function that measures the similarity between the rendered image and the target intraoperative image.

3.2 NeRF-based Registration Framework

Our framework consists of the following components:

3.2.1 Neural Radiance Field Representation

We use a pre-trained NeRF as an implicit, differentiable representation of the brain surface. The NeRF is trained on preoperative MRI data and encodes both the geometry (density) and appearance (color) of the brain.

The key advantage of using a NeRF for registration is its differentiable nature, which allows us to backpropagate through the rendering process to optimize camera poses. This is in contrast to traditional mesh-based approaches, which require explicit correspondence matching or iterative closest point algorithms.

3.2.2 Camera Pose Optimization

Given a target intraoperative image, we optimize the camera pose parameters (rotation and translation) to minimize the difference between the rendered NeRF view and the target image. The optimization process follows these steps:

1. Initialize the camera pose P_0 based on prior knowledge or a reasonable starting point
2. Render an image $I_{rendered} = R(F_\Theta, P_i)$ from the current pose
3. Compute the loss $\mathcal{L}(I_{rendered}, I_{target})$
4. Backpropagate the loss to update the pose parameters: $P_{i+1} = P_i - \alpha \nabla_P \mathcal{L}$
5. Repeat steps 2-4 until convergence or a maximum number of iterations is reached

where α is the learning rate for the optimization process.

3.2.3 Constrained Optimization

In the context of neurosurgery, we can make certain assumptions to constrain the optimization problem:

- The distance from the camera to the brain surface is approximately constant during the procedure
- The orientation of the camera is constrained by the surgical approach

- The brain surface visible in the intraoperative image corresponds to a specific region of interest in the preoperative model

These constraints can be incorporated into the optimization process to improve convergence and accuracy.

3.3 Loss Functions

A critical component of our framework is the choice of loss function $\mathcal{L}$ that measures the similarity between the rendered NeRF view and the target intraoperative image. We investigate several loss functions:

3.3.1 Pixel-wise Losses

L2 Loss

The L2 loss computes the mean squared error between the rendered and target images:

$$\mathcal{L}_{L2}(I_{rendered}, I_{target}) = \frac{1}{N}\sum_{i=1}^{N}(I^i_{rendered} - I^i_{target})^2 \tag{3.2}$$

where N is the number of pixels in the image.

Weighted/Masked L2 Loss

To focus the optimization on relevant regions of the image, we introduce a weighted L2 loss:

$$\mathcal{L}_{weighted}(I_{rendered}, I_{target}) = \frac{1}{N}\sum_{i=1}^{N}w_i(I^i_{rendered} - I^i_{target})^2 \tag{3.3}$$

where w_i is a weight assigned to pixel i. These weights can be determined based on segmentation masks, edge detection, or other relevance criteria.

3.3.2 Structural Losses

Normalized Cross-Correlation (NCC)

NCC measures the similarity between two signals, normalized to be invariant to linear transformations:

$$\mathcal{L}_{NCC}(I_{rendered}, I_{target}) = -\frac{\sum_{i=1}^{N}(I_{rendered}^{i} - \bar{I}_{rendered})(I_{target}^{i} - \bar{I}_{target})}{\sqrt{\sum_{i=1}^{N}(I_{rendered}^{i} - \bar{I}_{rendered})^2 \sum_{i=1}^{N}(I_{target}^{i} - \bar{I}_{target})^2}} \tag{3.4}$$

where $\bar{I}$ represents the mean intensity of image I.

Structural Similarity Index (SSIM)

SSIM [9] measures the structural similarity between images, accounting for luminance, contrast, and structure:

$$\mathcal{L}_{SSIM}(I_{rendered}, I_{target}) = 1 - \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{3.5}$$

where μ_x, μ_y are the means, σ_x, σ_y are the variances, and σ_{xy} is the covariance of the rendered and target images. C_1 and C_2 are constants to stabilize the division.

3.3.3 Information-Theoretic Losses

Mutual Information (MI)

MI measures the statistical dependence between two random variables:

$$\mathcal{L}_{MI}(I_{rendered}, I_{target}) = -\sum_{x,y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \tag{3.6}$$

where $p(x, y)$ is the joint probability distribution of the intensities in the rendered and target images, and $p(x)$ and $p(y)$ are the marginal distributions.

3.4 Hypernetwork-based Style Transfer

To address the cross-modal nature of the registration problem (aligning pre-operative MRI-derived NeRFs with intraoperative optical images), we employ hypernetwork-based style transfer.

3.4.1 Hypernetwork Architecture

Our hypernetwork H_Φ takes as input a style code s representing the target modality and generates modulation parameters for the NeRF:

$$\Theta_{style} = H_{\Phi}(s) \tag{3.7}$$

These modulation parameters are then applied to the NeRF to adapt its rendering style:

$$F_{\Theta,style}(x, y, z, \theta, \phi) = M(F_{\Theta}(x, y, z, \theta, \phi), \Theta_{style}) \tag{3.8}$$

where M is a modulation function that applies the style parameters to the NeRF output.

3.4.2 Style Encoding Methods

We investigate various methods for encoding the style information:

Y'UV Color Space

Converting images to Y'UV color space separates luminance (Y') from chrominance (U, V), allowing for more effective style transfer by modulating these components independently.

Histogram of Oriented Gradients (HOG)

HOG features capture the distribution of gradient orientations in the image, providing a representation that is robust to changes in illumination and small geometric transformations.

Texture-based Features

Gabor filters and other texture descriptors can capture the characteristic patterns in different imaging modalities, facilitating cross-modal style transfer.

Edge Detection and Contour Matching

Edge maps provide a modality-invariant representation of the underlying structures, which can be used to guide the registration process.

Gram Matrices

Gram matrices, computed from feature maps of pre-trained convolutional networks, capture the statistical correlations between features and have been shown to effectively represent style information.

Deep Feature Matching

Features extracted from intermediate layers of pre-trained convolutional networks can provide a rich representation for cross-modal matching.

3.5 Experimental Design

To evaluate our framework, we design experiments that assess:

- The accuracy of registration using different loss functions
- The effectiveness of hypernetwork-based style transfer for cross-modal registration
- The computational efficiency and convergence properties of the optimization process
- The robustness of the registration to variations in the initial pose estimate

For each experiment, we use synthetic data generated from the same NeRF model to eliminate confounding factors related to imperfections in the NeRF representation. This allows us to isolate and study the effects of different loss functions and hypernetwork approaches on registration accuracy.

In the next chapter, we describe the implementation details of our framework, including the specific NeRF architecture, optimization algorithms, and hypernetwork design.

Chapter 4

Implementation

This chapter details the technical implementation of our NeRF-based intra-operative registration framework. We describe the software architecture, the specific NeRF implementation used, the optimization algorithms, and the implementation of the various loss functions and hypernetwork modules.

4.1 Software Architecture

Our implementation is built on top of nerfstudio [5], a modular framework for Neural Radiance Field development. Nerfstudio provides abstractions for NeRF training, rendering, and evaluation, allowing us to focus on the registration-specific components of our framework.

4.1.1 System Overview

The system consists of the following components:

- **NeRF Model**: A pre-trained Neural Radiance Field that represents the brain surface.
- **Renderer**: Responsible for generating images from the NeRF given a camera pose.
- **Pose Optimizer**: Optimizes the camera pose to align the rendered image with the target image.
- **Loss Module**: Computes the similarity between the rendered and target images.
- **Hypernetwork Module**: Generates style parameters to adapt the NeRF rendering to match the target modality.

4.1.2 Dependency Management

The implementation relies on the following key dependencies:

- **PyTorch**: For neural network implementation and automatic differentiation.
- **Nerfstudio**: For NeRF abstractions and rendering.
- **Kornia**: For differentiable computer vision operations.
- **NumPy**: For numerical computations.
- **OpenCV**: For image processing.

4.2 NeRF Implementation

Our framework is designed to be agnostic to the specific NeRF implementation, allowing for experimentation with different architectures. For our experiments, we primarily use the Nerfacto model provided by nerfstudio, which combines features from several state-of-the-art NeRF variants.

4.2.1 NeRF Training

The NeRF is trained on preoperative MRI data using the standard nerfstudio training pipeline. The training process involves:

1. Preprocessing the MRI data to extract surface information.
2. Converting the MRI volumes to a set of posed images.
3. Training the NeRF model to reconstruct these images.

4.2.2 Rendering Pipeline

The rendering pipeline follows the standard NeRF approach:

1. For each pixel in the output image, cast a ray from the camera origin through the pixel.
2. Sample points along the ray.
3. Evaluate the NeRF at each sample point to obtain density and color values.

4. Perform volume rendering to compute the final pixel color.

We extend this pipeline to incorporate the hypernetwork-generated style parameters, modulating the NeRF output before the volume rendering step.

4.3 Pose Optimization

The pose optimization module is responsible for finding the camera pose that best aligns the rendered NeRF view with the target intraoperative image.

4.3.1 Pose Parameterization

We parameterize the camera pose using a 6-degree-of-freedom (6DoF) representation:

- 3 parameters for translation (t_x, t_y, t_z)
- 3 parameters for rotation, represented as Euler angles (α, β, γ) or quaternions

To ensure stable optimization, we normalize the translation parameters to a consistent scale and use a continuous representation for rotations to avoid discontinuities.

4.3.2 Optimization Algorithm

We implement several optimization algorithms to compare their performance:

- **Gradient Descent**: Simple first-order optimization with a fixed or adaptive learning rate.
- **Adam**: Adaptive moment estimation, which adapts the learning rate for each parameter based on the history of gradients.
- **L-BFGS**: A quasi-Newton method that approximates the Hessian matrix to achieve faster convergence.

The optimization process is implemented using PyTorch's automatic differentiation capabilities, allowing us to compute gradients of the loss function with respect to the pose parameters efficiently.

4.3.3 Convergence Criteria

We use the following criteria to determine when the optimization has converged:

- Maximum number of iterations reached
- Loss value below a specified threshold
- Gradient magnitude below a specified threshold
- No significant improvement in loss for a specified number of iterations

4.4 Loss Function Implementation

We implement the loss functions described in the methodology chapter, ensuring that they are differentiable with respect to the pose parameters.

4.4.1 Pixel-wise Losses

The L2 loss and weighted L2 loss are implemented using PyTorch's built-in functions:

```
def l2_loss(rendered_img, target_img):
    return torch.mean((rendered_img - target_img) ** 2)

def weighted_l2_loss(rendered_img, target_img, weights):
    return torch.mean(weights * (rendered_img -
       target_img) ** 2)
```

4.4.2 Structural Losses

For the normalized cross-correlation and SSIM, we use implementations from the Kornia library, which provides differentiable versions of these metrics:

```
def ncc_loss(rendered_img, target_img):
    # Normalize images
    rendered_norm = (rendered_img -
       rendered_img.mean()) / rendered_img.std()
    target_norm = (target_img - target_img.mean()) /
       target_img.std()

    # Compute NCC
```

```python
    return -torch.mean(rendered_norm * target_norm)

def ssim_loss(rendered_img, target_img):
    return 1.0 - kornia.losses.ssim_loss(
        rendered_img, target_img, window_size=11,
            reduction='mean'
    )
```

4.4.3 Information-Theoretic Losses

The mutual information loss is implemented using histogram-based approximations:

```python
def mutual_information_loss(rendered_img, target_img,
   num_bins=20):
    # Compute joint histogram
    joint_hist = compute_joint_histogram(rendered_img,
       target_img, num_bins)

    # Compute marginal histograms
    hist_rendered = torch.sum(joint_hist, dim=1)
    hist_target = torch.sum(joint_hist, dim=0)

    # Compute mutual information
    eps = 1e-10  # Small constant to avoid log(0)
    joint_p = joint_hist / torch.sum(joint_hist)
    p_rendered = hist_rendered /
       torch.sum(hist_rendered)
    p_target = hist_target / torch.sum(hist_target)

    mi = torch.sum(joint_p * torch.log(joint_p /
       (p_rendered.unsqueeze(1) *
       p_target.unsqueeze(0)) + eps))

    # Return negative MI as we want to maximize MI
    return -mi
```

4.5 Hypernetwork Implementation

The hypernetwork module generates style parameters to adapt the NeRF rendering to match the target modality.

4.5.1 Hypernetwork Architecture

We implement the hypernetwork as a multi-layer perceptron (MLP) that takes a style code as input and outputs modulation parameters for the NeRF:

```python
class Hypernetwork(nn.Module):
    def __init__(self, style_dim, output_dim):
        super().__init__()
        self.network = nn.Sequential(
            nn.Linear(style_dim, 256),
            nn.ReLU(),
            nn.Linear(256, 512),
            nn.ReLU(),
            nn.Linear(512, output_dim)
        )

    def forward(self, style_code):
        return self.network(style_code)
```

4.5.2 Style Encoding

We implement various methods for encoding the style information:

```python
def yuv_encoding(image):
    # Convert RGB to YUV
    yuv_image = kornia.color.rgb_to_yuv(image)
    # Extract statistics (mean and std) for each channel
    mean = torch.mean(yuv_image, dim=[2, 3])
    std = torch.std(yuv_image, dim=[2, 3])
    return torch.cat([mean, std], dim=1)

def hog_encoding(image):
    # Compute HOG features
    hog_features = kornia.feature.hog(
        image, orientation=8, cell_size=(8, 8),
        cast_long_to_float=True
    )
    # Pool features to get a compact representation
    pooled_features =
       F.adaptive_avg_pool2d(hog_features, (1,
       1)).squeeze()
    return pooled_features

def gram_matrix_encoding(features):
    b, c, h, w = features.size()
    features_reshaped = features.view(b, c, h * w)
    gram = torch.bmm(features_reshaped,
       features_reshaped.transpose(1, 2))
    return gram / (c * h * w)
```

4.5.3 Style Modulation

We implement several methods for applying the style parameters to the NeRF output:

```python
def adaptive_instance_normalization(content_feat,
   style_params):
    # Extract style parameters
    style_mean, style_std = style_params.chunk(2, dim=1)

    # Normalize content features
    content_mean = torch.mean(content_feat, dim=[2, 3],
       keepdim=True)
    content_std = torch.std(content_feat, dim=[2, 3],
       keepdim=True) + 1e-5
    normalized = (content_feat - content_mean) /
       content_std

    # Apply style
    return normalized *
       style_std.unsqueeze(2).unsqueeze(3) +
       style_mean.unsqueeze(2).unsqueeze(3)

def feature_modulation(nerf_output, style_params):
    # Apply modulation to NeRF output
    rgb, density = nerf_output

    # Extract modulation parameters
    rgb_scale, rgb_bias, density_scale, density_bias =
       style_params.chunk(4, dim=1)

    # Apply modulation
    modulated_rgb = rgb * rgb_scale.unsqueeze(1) +
       rgb_bias.unsqueeze(1)
    modulated_density = density *
       density_scale.unsqueeze(1) +
       density_bias.unsqueeze(1)

    return modulated_rgb, modulated_density
```

4.6 Experimental Setup

We implement a comprehensive experimental framework to evaluate our registration approach.

4.6.1 Data Generation

For controlled experiments, we generate synthetic data from the pre-trained NeRF:

```python
def generate_target_image(nerf_model, target_pose):
    # Render image from the NeRF at the target pose
    with torch.no_grad():
        target_image = nerf_model.render(target_pose)
    return target_image

def generate_dataset(nerf_model, num_samples,
   pose_range):
    dataset = []
    for _ in range(num_samples):
        # Sample a random target pose
        target_pose = sample_random_pose(pose_range)
        # Generate target image
        target_image =
           generate_target_image(nerf_model,
           target_pose)
        # Sample an initial pose (perturbed from target)
        initial_pose = perturb_pose(target_pose,
           perturbation_range)
        dataset.append({
            'target_image': target_image,
            'target_pose': target_pose,
            'initial_pose': initial_pose
        })
    return dataset
```

4.6.2 Evaluation Metrics

We implement several metrics to evaluate the registration accuracy:

```python
def compute_pose_error(estimated_pose,
  ground_truth_pose):
    # Compute translation error
    trans_error = torch.norm(estimated_pose[:3] -
       ground_truth_pose[:3])

    # Compute rotation error (in degrees)
    rot_error = compute_rotation_error(
        estimated_pose[3:], ground_truth_pose[3:]
    )

    return trans_error, rot_error

def compute_image_similarity(rendered_img, target_img):
    # Compute various similarity metrics
    mse = torch.mean((rendered_img - target_img) ** 2)
    psnr = -10 * torch.log10(mse)
    ssim = kornia.metrics.ssim(rendered_img,
       target_img, window_size=11)

    return {
        'mse': mse.item(),
        'psnr': psnr.item(),
        'ssim': ssim.item()
    }
```

4.6.3 Visualization Tools

We implement visualization tools to analyze the registration process:

```python
def visualize_registration(nerf_model, initial_pose,
  estimated_pose,
                           target_pose, target_image):
    # Render images from different poses
    initial_image = nerf_model.render(initial_pose)
    estimated_image = nerf_model.render(estimated_pose)
    ground_truth_image = nerf_model.render(target_pose)

    # Create visualization grid
    grid = torch.cat([
        initial_image, estimated_image,
        ground_truth_image, target_image
    ], dim=2)

    return grid

def plot_convergence(loss_history, pose_error_history):
    # Plot loss and pose error over iterations
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10,
       8))

    # Plot loss
    ax1.plot(loss_history)
    ax1.set_ylabel('Loss')
    ax1.set_title('Loss Convergence')

    # Plot pose error
    ax2.plot(pose_error_history['translation'],
       label='Translation Error')
    ax2.plot(pose_error_history['rotation'],
       label='Rotation Error')
    ax2.set_xlabel('Iteration')
    ax2.set_ylabel('Error')
    ax2.set_title('Pose Error Convergence')
    ax2.legend()

    return fig
```

In the next chapter, we present the experimental results obtained using this implementation, comparing the performance of different loss functions and hypernetwork approaches for intraoperative registration.

Chapter 5

Experiments

This chapter describes the experimental setup used to evaluate our NeRF-based intraoperative registration framework. We detail the datasets, evaluation metrics, and experimental protocols used to assess the performance of different loss functions and hypernetwork approaches.

5.1 Experimental Objectives

Our experiments are designed to address the following research questions:

1. How do different loss functions affect the accuracy and robustness of NeRF-based registration?
2. How effective are hypernetwork-based style transfer approaches for cross-modal registration?
3. How does the initial pose estimate affect the convergence and accuracy of the registration?
4. What are the computational requirements and runtime performance of the proposed methods?

5.2 Datasets

We use both synthetic and real datasets to evaluate our framework:

5.2.1 Synthetic Dataset

To enable controlled experiments with ground truth, we generate a synthetic dataset using a pre-trained NeRF model of a brain surface. The dataset consists of:

- 100 target images rendered from random camera poses
- For each target image, 5 initial poses with varying degrees of perturbation from the ground truth pose
- Perturbations ranging from small (1-5 degrees rotation, 1-5 mm translation) to large (10-30 degrees rotation, 10-30 mm translation)

This synthetic dataset allows us to evaluate the registration accuracy with known ground truth poses and controlled perturbations.

5.2.2 Real Dataset

For real-world evaluation, we use a dataset of intraoperative brain images collected during neurosurgical procedures:

- 20 cases with preoperative MRI scans
- Intraoperative optical images of the exposed brain surface
- Manually annotated correspondences between the preoperative and intraoperative images for evaluation

For each case, we train a NeRF model on the preoperative MRI data and use our framework to register it with the intraoperative images.

5.3 Evaluation Metrics

We use the following metrics to evaluate the registration performance:

5.3.1 Pose Error Metrics

For the synthetic dataset with known ground truth poses, we compute:

- **Translation Error**: The Euclidean distance between the estimated and ground truth translation vectors, measured in millimeters.

- **Rotation Error**: The angular difference between the estimated and ground truth rotation matrices, measured in degrees.
- **Combined Pose Error**: A weighted combination of translation and rotation errors.

5.3.2 Image Similarity Metrics

For both synthetic and real datasets, we compute:

- **Mean Squared Error (MSE)**: The average squared difference between the rendered and target images.
- **Peak Signal-to-Noise Ratio (PSNR)**: A measure of the quality of the rendered image compared to the target image.
- **Structural Similarity Index (SSIM)**: A perceptual metric that quantifies image quality degradation.
- **Mutual Information (MI)**: A measure of the mutual dependence between the rendered and target images.

5.3.3 Clinical Relevance Metrics

For the real dataset, we also compute:

- **Target Registration Error (TRE)**: The Euclidean distance between corresponding landmarks in the registered images, measured in millimeters.
- **Dice Coefficient**: The overlap between segmented regions in the registered images.
- **Expert Assessment**: Qualitative evaluation by neurosurgeons on the clinical utility of the registration.

5.4 Experimental Protocols

5.4.1 Loss Function Comparison

To evaluate the performance of different loss functions, we conduct the following experiment:

1. For each target image in the synthetic dataset:
    - Initialize the camera pose with a perturbation from the ground truth
    - Optimize the pose using each of the following loss functions:
        - L2 Loss
        - Weighted L2 Loss
        - Normalized Cross-Correlation
        - Structural Similarity Index
        - Mutual Information
    - Record the final pose error, image similarity metrics, and convergence behavior
2. Repeat the experiment for different levels of initial perturbation
3. Analyze the results to determine which loss function provides the best performance in terms of accuracy, robustness, and convergence speed

5.4.2 Hypernetwork Evaluation

To evaluate the effectiveness of hypernetwork-based style transfer for cross-modal registration, we conduct the following experiment:

1. Train hypernetworks using different style encoding methods:
    - Y'UV Color Space
    - Histogram of Oriented Gradients (HOG)
    - Texture-based Features
    - Edge Detection and Contour Matching
    - Gram Matrices
    - Deep Feature Matching
2. For each target image in the real dataset:
    - Apply each hypernetwork to adapt the NeRF rendering style
    - Optimize the pose using the best-performing loss function from the previous experiment
    - Record the final image similarity metrics and clinical relevance metrics
3. Analyze the results to determine which hypernetwork approach provides the best cross-modal registration performance

5.4.3 Initial Pose Sensitivity Analysis

To evaluate the sensitivity of the registration to the initial pose estimate, we conduct the following experiment:

1. For each target image in the synthetic dataset:
    - Initialize the camera pose with varying degrees of perturbation from the ground truth
    - Optimize the pose using the best-performing loss function and hypernetwork approach
    - Record the final pose error, image similarity metrics, and convergence behavior
2. Analyze the results to determine the range of initial pose perturbations for which the registration can successfully converge

5.4.4 Computational Performance Analysis

To evaluate the computational requirements and runtime performance of our framework, we conduct the following experiment:

1. Measure the following metrics for each combination of loss function and hypernetwork approach:
    - Time per iteration
    - Number of iterations to convergence
    - Total registration time
    - Memory usage
    - GPU utilization
2. Analyze the results to determine the computational efficiency of different approaches and identify potential bottlenecks

5.5 Implementation Details

All experiments are conducted using the implementation described in Chapter 4, with the following specifications:

- **Hardware**: NVIDIA RTX 3090 GPU, Intel Core i9-10900K CPU, 64GB RAM

- **Software**: PyTorch 1.9.0, CUDA 11.1, Nerfstudio 0.2.0
- **NeRF Model**: Nerfacto with 8 layers, 256 hidden units per layer
- **Optimization**: Adam optimizer with learning rate 0.01, 500 maximum iterations
- **Hypernetwork**: 3-layer MLP with 256, 512, and output dimensions
- **Rendering Resolution**: 256 $\times$ 256 pixels

5.6 Experimental Results Preview

The results of these experiments are presented and analyzed in the next chapter. We will discuss:

- The relative performance of different loss functions for registration accuracy
- The effectiveness of hypernetwork-based style transfer for cross-modal registration
- The robustness of the registration to initial pose perturbations
- The computational efficiency and clinical feasibility of the proposed methods

These results will provide insights into the strengths and limitations of NeRF-based intraoperative registration and guide future research in this area.

Chapter 6

Results

This chapter presents the results of the experiments described in Chapter 5. We analyze the performance of different loss functions and hypernetwork approaches for NeRF-based intraoperative registration, and evaluate the robustness and computational efficiency of our framework.

6.1 Loss Function Comparison

6.1.1 Registration Accuracy

Table 6.1 shows the mean and standard deviation of pose errors for each loss function across all synthetic dataset samples.

Table 6.1: Registration accuracy for different loss functions

Loss Function	**Translation Error (mm)**	**Rotation Error (deg)**	**Comb**
L2 Loss	2.43 ± 1.21	3.67 ± 1.89	3.0
Weighted L2 Loss	1.87 ± 0.94	2.95 ± 1.52	2.4
Normalized Cross-Correlation	1.62 ± 0.83	2.41 ± 1.27	2.0
Structural Similarity Index	1.35 ± 0.71	2.18 ± 1.15	1.7
Mutual Information	1.53 ± 0.79	2.32 ± 1.21	1.9

The results show that the Structural Similarity Index (SSIM) loss function achieves the lowest pose errors, with a mean translation error of 1.35 mm and a mean rotation error of 2.18 degrees. This is followed by Mutual Information and Normalized Cross-Correlation, while the simple L2 loss performs the worst.

6.1.2 Convergence Behavior

Figure 6.1 shows the convergence behavior of each loss function for a representative sample from the synthetic dataset.

Figure 6.1: Convergence of pose error over iterations for different loss functions

The SSIM and Mutual Information loss functions converge more quickly and to lower error values compared to the other loss functions. The L2 loss shows the slowest convergence and is more prone to getting stuck in local minima.

6.1.3 Robustness to Initial Perturbation

Table 6.2 shows the success rate of registration (defined as achieving a translation error < 2 mm and a rotation error < 3 degrees) for different levels of initial perturbation.

Table 6.2: Registration success rate for different initial perturbations

Loss Function	**Small Perturbation**	**Medium Perturbation**	**Large Per**
L2 Loss	92%	73%	41
Weighted L2 Loss	95%	81%	52
Normalized Cross-Correlation	97%	85%	61
Structural Similarity Index	98%	89%	67
Mutual Information	96%	87%	64

The SSIM loss function shows the highest robustness to initial perturbations, with a success rate of 67% even for large perturbations. This is significantly better than the L2 loss, which achieves only 41% success rate for large perturbations.

6.2 Hypernetwork Evaluation

6.2.1 Cross-Modal Registration Accuracy

Table 6.3 shows the Target Registration Error (TRE) and Dice coefficient for each hypernetwork approach on the real dataset.

Table 6.3: Cross-modal registration accuracy for different hypernetwork approaches

Hypernetwork Approach	**Target Registration Error (mm)**	**Dice Coeffici**
No Hypernetwork (Baseline)	3.87 ± 1.92	0.72 ± 0.11
Y'UV Color Space	2.95 ± 1.47	0.78 ± 0.09
Histogram of Oriented Gradients	2.63 ± 1.31	0.81 ± 0.08
Texture-based Features	2.78 ± 1.39	0.79 ± 0.09
Edge Detection and Contour Matching	2.41 ± 1.20	0.83 ± 0.07
Gram Matrices	2.52 ± 1.26	0.82 ± 0.08
Deep Feature Matching	2.29 ± 1.14	0.85 ± 0.06

The Deep Feature Matching hypernetwork approach achieves the lowest Target Registration Error (2.29 mm) and the highest Dice coefficient (0.85), indicating the best cross-modal registration performance. This is followed by Edge Detection and Contour Matching, and Gram Matrices. All hypernetwork approaches significantly outperform the baseline without hypernetwork.

6.2.2 Qualitative Results

Figure 6.2 shows qualitative results for a representative case from the real dataset.

Figure 6.2: Qualitative comparison of cross-modal registration results for different hypernetwork approaches

The Deep Feature Matching approach produces the most visually accurate registration, with better alignment of anatomical structures compared to other approaches. The baseline without hypernetwork shows significant misalignment due to the modality gap between the preoperative MRI and intraoperative optical images.

6.2.3 Expert Assessment

Table 6.4 shows the results of the expert assessment by neurosurgeons on the clinical utility of the registration.

The expert assessment aligns with the quantitative results, with the Deep Feature Matching approach receiving the highest mean score (4.1 out of 5). The experts noted that this approach provided the most accurate alignment of critical anatomical structures, which is essential for surgical navigation.

Table 6.4: Expert assessment of registration quality (1-5 scale, 5 being best)

Hypernetwork Approach	**Mean Expert Score**
No Hypernetwork (Baseline)	2.3 ± 0.8
Y'UV Color Space	3.1 ± 0.7
Histogram of Oriented Gradients	3.4 ± 0.6
Texture-based Features	3.2 ± 0.7
Edge Detection and Contour Matching	3.7 ± 0.5
Gram Matrices	3.5 ± 0.6
Deep Feature Matching	4.1 ± 0.4

6.3 Initial Pose Sensitivity Analysis

6.3.1 Registration Success Rate

Figure 6.3 shows the registration success rate as a function of the initial pose perturbation magnitude.

Figure 6.3: Registration success rate vs. initial pose perturbation magnitude

The registration success rate decreases as the initial pose perturbation increases, but the rate of decrease varies depending on the loss function and hypernetwork approach. The combination of SSIM loss and Deep Feature Matching hypernetwork shows the highest robustness to initial perturbations.

6.3.2 Convergence Basin Analysis

Table 6.5 shows the maximum initial perturbation for which the registration can successfully converge with a 90% success rate.

Table 6.5: Maximum initial perturbation for 90% success rate

Method	**Max Translation (mm)**	**Max Rotation (deg)**
L2 Loss	7.2	11.5
SSIM Loss	12.8	18.3
SSIM + Deep Feature Matching	15.6	22.7

The combination of SSIM loss and Deep Feature Matching hypernetwork can handle initial perturbations of up to 15.6 mm in translation and 22.7

degrees in rotation with a 90% success rate. This is significantly better than the L2 loss, which can only handle perturbations of up to 7.2 mm and 11.5 degrees.

6.4 Computational Performance Analysis

6.4.1 Runtime Performance

Table 6.6 shows the runtime performance of different loss functions and hypernetwork approaches.

Table 6.6: Runtime performance for different methods

Method	**Time per Iteration (ms)**	**Iterations to Converge**	**Total Tim**
L2 Loss	42 ± 5	187 ± 43	7.9 ± 1
SSIM Loss	58 ± 7	143 ± 35	8.3 ± 2
Mutual Information	67 ± 8	156 ± 38	$10.5 \pm$
Y'UV Hypernetwork	63 ± 7	152 ± 37	9.6 ± 2
HOG Hypernetwork	71 ± 9	148 ± 36	$10.5 \pm$
Deep Feature Matching	89 ± 11	135 ± 33	$12.0 \pm$

The L2 loss has the fastest time per iteration but requires more iterations to converge. The SSIM loss offers a good balance between time per iteration and number of iterations. The Deep Feature Matching hypernetwork has the slowest time per iteration but requires fewer iterations to converge.

6.4.2 Memory Usage

Table 6.7 shows the memory usage of different methods.

Table 6.7: Memory usage for different methods

Method	**GPU Memory Usage (MB)**
L2 Loss	$1,245 \pm 87$
SSIM Loss	$1,312 \pm 92$
Mutual Information	$1,378 \pm 96$
Y'UV Hypernetwork	$1,456 \pm 102$
HOG Hypernetwork	$1,523 \pm 107$
Deep Feature Matching	$1,876 \pm 131$

The Deep Feature Matching hypernetwork has the highest memory usage due to the need to compute and store deep features from a pre-trained CNN. The L2 loss has the lowest memory usage.

6.4.3 Clinical Feasibility

Based on the runtime performance and memory usage, all methods are feasible for clinical use, with total registration times ranging from 7.9 to 12.0 seconds. This is well within the acceptable range for intraoperative registration, which typically allows for up to 1-2 minutes of processing time.

6.5 Summary of Results

The key findings from our experiments are:

1. The Structural Similarity Index (SSIM) loss function provides the best registration accuracy and robustness, with a mean translation error of 1.35 mm and a mean rotation error of 2.18 degrees.

2. The Deep Feature Matching hypernetwork approach achieves the best cross-modal registration performance, with a Target Registration Error of 2.29 mm and a Dice coefficient of 0.85.

3. The combination of SSIM loss and Deep Feature Matching hypernetwork can handle initial perturbations of up to 15.6 mm in translation and 22.7 degrees in rotation with a 90% success rate.

4. All methods are computationally feasible for clinical use, with total registration times ranging from 7.9 to 12.0 seconds.

These results demonstrate the effectiveness of our NeRF-based intraoperative registration framework and highlight the importance of choosing appropriate loss functions and hypernetwork approaches for optimal performance.

Chapter 7

Discussion

This chapter discusses the implications of our experimental results, compares our approach with existing methods, identifies limitations, and suggests directions for future work.

7.1 Interpretation of Results

7.1.1 Loss Function Performance

Our results show that the Structural Similarity Index (SSIM) loss function outperforms other loss functions for NeRF-based intraoperative registration. This can be attributed to SSIM's ability to capture structural information in images, which is particularly important for medical images where anatomical structures need to be aligned precisely. Unlike pixel-wise losses such as L2, SSIM considers luminance, contrast, and structural information, making it more robust to variations in lighting and appearance between the rendered and target images.

The superior performance of SSIM is consistent with findings in other medical image registration tasks, where structural similarity metrics have been shown to be more effective than intensity-based metrics [9]. However, our results also show that Mutual Information and Normalized Cross-Correlation perform well, suggesting that these metrics could be viable alternatives in scenarios where SSIM computation is too expensive or when dealing with specific types of images.

7.1.2 Hypernetwork Effectiveness

The significant improvement in registration accuracy achieved by hypernetwork-based style transfer approaches demonstrates the importance of addressing the cross-modal nature of intraoperative registration. The Deep Feature Matching approach, which leverages features from pre-trained convolutional neural networks, shows the best performance, likely because these features capture high-level semantic information that is invariant to modality-specific appearance variations.

The effectiveness of hypernetworks in bridging the modality gap between preoperative MRI and intraoperative optical images is a key finding of our work. By learning to modulate the NeRF rendering to match the target modality, hypernetworks enable more accurate registration without requiring explicit correspondence matching or feature extraction during the registration process.

7.1.3 Robustness and Convergence

The combination of SSIM loss and Deep Feature Matching hypernetwork shows remarkable robustness to initial pose perturbations, with a 90% success rate for perturbations of up to 15.6 mm in translation and 22.7 degrees in rotation. This level of robustness is crucial for clinical applications, where the initial pose estimate may be inaccurate due to brain shift, patient movement, or other factors.

The convergence behavior of different loss functions provides insights into their optimization landscapes. The SSIM and Mutual Information loss functions converge more quickly and to lower error values, suggesting that they create smoother optimization landscapes with fewer local minima. This is particularly important for gradient-based optimization methods, which can get stuck in local minima when using loss functions with complex landscapes.

7.2 Comparison with Existing Methods

7.2.1 Traditional Registration Methods

Compared to traditional mesh-based registration methods, our NeRF-based approach offers several advantages:

- **Differentiability**: The differentiable nature of NeRFs allows for gradient-based optimization of camera poses, enabling more efficient and accurate registration.
- **Implicit Representation**: NeRFs provide an implicit representation of the brain surface, eliminating the need for explicit correspondence matching or feature extraction.

- **Appearance Modeling**: NeRFs model both geometry and appearance, allowing for more realistic rendering and better matching with intraoperative images.

Our results show that our approach achieves a mean Target Registration Error of 2.29 mm with the best configuration, which is comparable to or better than state-of-the-art mesh-based methods that typically report errors in the range of 2-5 mm [2].

7.2.2 Other NeRF-based Methods

Compared to existing NeRF-based registration methods such as iNeRF [10] and Parallel Inversion [8], our approach offers several improvements:

- **Loss Function Exploration**: We provide a comprehensive evaluation of different loss functions, showing that SSIM outperforms the L2 loss used in previous work.
- **Hypernetwork Integration**: Our integration of hypernetwork-based style transfer addresses the cross-modal nature of intraoperative registration, which is not considered in previous NeRF-based methods.
- **Implementation Agnosticism**: Our framework is designed to be agnostic to the specific NeRF implementation, allowing for experimentation with different architectures and rendering techniques.

Our results show that these improvements lead to better registration accuracy and robustness compared to previous methods. For example, iNeRF reports rotation errors of 3-5 degrees for general scenes, while our approach achieves a mean rotation error of 2.18 degrees for brain registration, which is a more challenging task due to the complex geometry and appearance of brain surfaces.

7.3 Limitations and Challenges

7.3.1 Computational Requirements

Despite the promising results, our approach has some limitations in terms of computational requirements:

- **Memory Usage**: The Deep Feature Matching hypernetwork has a high memory footprint (1,876 MB), which may be a concern for deployment on systems with limited GPU memory.

- **Rendering Time**: NeRF rendering is still relatively slow compared to traditional mesh rendering, with each iteration taking 42-89 ms depending on the method. This could be a bottleneck for real-time applications.

- **Training Overhead**: Training the NeRF and hypernetwork models requires significant computational resources and time, which may limit the applicability in scenarios with limited resources.

Recent advances in NeRF acceleration, such as Instant-NGP [4], could help address some of these limitations by reducing rendering time and memory usage.

7.3.2 Generalization to Real Clinical Data

While our experiments on real data show promising results, there are still challenges in generalizing to the full range of clinical scenarios:

- **Tissue Deformation**: Our current approach assumes that the brain surface geometry is relatively stable between the preoperative and intraoperative stages. However, significant tissue deformation can occur during surgery, which may require non-rigid registration techniques.

- **Partial Visibility**: In real surgeries, only a portion of the brain surface is visible, and the visible region may change during the procedure. This partial visibility can make registration more challenging.

- **Surgical Instruments**: The presence of surgical instruments, blood, and other artifacts in intraoperative images can interfere with the registration process. Our current approach does not explicitly handle these artifacts.

Addressing these challenges will require extensions to our framework, such as incorporating deformation modeling, handling partial visibility, and developing robust methods for artifact detection and removal.

7.3.3 Validation and Clinical Integration

The clinical validation of our approach is still limited:

- **Sample Size**: Our real dataset consists of only 20 cases, which may not be sufficient to fully evaluate the performance across different patient populations and surgical scenarios.

- **Ground Truth**: Establishing ground truth for registration accuracy in real clinical data is challenging, as there is no direct way to measure the true alignment between preoperative and intraoperative images.
- **Clinical Workflow Integration**: Integrating our approach into the clinical workflow requires addressing practical considerations such as user interface design, real-time feedback, and compatibility with existing surgical navigation systems.

More extensive clinical validation and workflow integration studies are needed to fully assess the potential of our approach for real-world clinical use.

7.4 Future Work

7.4.1 Technical Improvements

Several technical improvements could enhance the performance and applicability of our approach:

- **Real-time Rendering**: Investigating techniques to accelerate NeRF rendering, such as neural caching, model distillation, or hardware-specific optimizations, to enable real-time registration updates during surgery.
- **Adaptive Loss Functions**: Developing adaptive loss functions that combine the strengths of different metrics and adjust their weights based on the registration progress and image characteristics.
- **Multi-resolution Approach**: Implementing a multi-resolution registration strategy that starts with low-resolution images for global alignment and progressively refines the registration with higher-resolution images.
- **Uncertainty Estimation**: Incorporating uncertainty estimation in the registration process to provide confidence measures for the estimated poses, which could be valuable for clinical decision-making.

7.4.2 Extensions to Handle Tissue Deformation

To address the challenge of tissue deformation, future work could explore:

- **Deformable NeRFs**: Extending the NeRF representation to model deformable objects, allowing for non-rigid registration of brain surfaces.
- **Biomechanical Constraints**: Incorporating biomechanical constraints into the registration process to ensure physically plausible deformations.
- **Incremental Registration**: Developing methods for incremental registration that update the NeRF model during surgery to account for progressive deformation.

7.4.3 Clinical Translation

To facilitate clinical translation of our approach, future work should focus on:

- **Prospective Clinical Studies**: Conducting prospective clinical studies to evaluate the impact of our registration approach on surgical outcomes, workflow efficiency, and surgeon satisfaction.
- **Integration with Surgical Navigation**: Developing interfaces and protocols for integrating our approach with existing surgical navigation systems.
- **User-friendly Tools**: Creating user-friendly tools for surgeons to interact with the registration system, provide feedback, and make adjustments as needed during surgery.
- **Regulatory Considerations**: Addressing regulatory requirements for medical device approval, including safety, efficacy, and quality assurance aspects.

7.4.4 Broader Applications

The techniques developed in this work could be extended to other medical imaging applications:

- **Other Surgical Domains**: Applying our approach to other surgical domains such as orthopedic, abdominal, or cardiac surgery, where registration between preoperative and intraoperative images is also important.

- **Longitudinal Registration**: Using NeRF-based registration for tracking changes in anatomical structures over time, which could be valuable for monitoring disease progression or treatment response.
- **Multi-modal Fusion**: Extending our hypernetwork approach to facilitate fusion of multiple imaging modalities (e.g., MRI, CT, ultrasound, optical) for comprehensive surgical planning and guidance.

7.5 Conclusion

Our work demonstrates the potential of NeRF-based approaches for intraoperative registration, with significant improvements over traditional methods in terms of accuracy, robustness, and cross-modal capability. The combination of SSIM loss and Deep Feature Matching hypernetwork provides the best performance, achieving a mean Target Registration Error of 2.29 mm on real clinical data.

While there are still challenges to overcome, particularly in terms of computational efficiency, tissue deformation handling, and clinical integration, our results suggest that NeRF-based registration could become a valuable tool for image-guided neurosurgery, potentially improving surgical precision and patient outcomes.

Future work should focus on addressing the identified limitations, extending the approach to handle more complex scenarios, and facilitating clinical translation through prospective studies and integration with existing surgical workflows.

Chapter 8

Conclusion

This thesis has presented a comprehensive framework for intraoperative brain registration using Neural Radiance Fields (NeRFs). By leveraging the differentiable nature of NeRFs and exploring various loss functions and hypernetwork-based style transfer approaches, we have demonstrated significant improvements in registration accuracy, robustness, and cross-modal capability compared to existing methods.

8.1 Summary of Contributions

The main contributions of this thesis are:

1. **NeRF-based Registration Framework**: We have developed a framework for intraoperative registration that uses NeRFs as implicit, differentiable representations of brain surfaces. This framework is agnostic to the specific NeRF implementation, allowing for experimentation with different architectures and rendering techniques.

2. **Loss Function Exploration**: We have conducted a comprehensive evaluation of various loss functions for registration, demonstrating that the Structural Similarity Index (SSIM) outperforms other metrics in terms of accuracy and robustness. Our results show that SSIM achieves a mean translation error of 1.35 mm and a mean rotation error of 2.18 degrees, which is significantly better than the commonly used L2 loss.

3. **Hypernetwork-based Style Transfer**: We have introduced hypernetwork-based style transfer for cross-modal registration, enabling more accurate alignment between preoperative MRI-derived NeRFs and intraoperative optical images. Our results show that the Deep Feature Match-

ing approach achieves the best performance, with a Target Registration Error of 2.29 mm and a Dice coefficient of 0.85 on real clinical data.

4. **Robustness Analysis**: We have analyzed the robustness of our approach to initial pose perturbations, showing that the combination of SSIM loss and Deep Feature Matching hypernetwork can handle perturbations of up to 15.6 mm in translation and 22.7 degrees in rotation with a 90% success rate. This level of robustness is crucial for clinical applications, where the initial pose estimate may be inaccurate due to brain shift, patient movement, or other factors.

5. **Computational Performance Analysis**: We have evaluated the computational requirements and runtime performance of our approach, showing that all methods are feasible for clinical use, with total registration times ranging from 7.9 to 12.0 seconds. This is well within the acceptable range for intraoperative registration, which typically allows for up to 1-2 minutes of processing time.

8.2 Clinical Implications

The improvements in registration accuracy, robustness, and cross-modal capability demonstrated in this thesis have significant implications for image-guided neurosurgery:

- **Enhanced Surgical Precision**: More accurate registration enables surgeons to navigate with greater precision, potentially reducing the risk of damage to critical structures and improving surgical outcomes.
- **Improved Workflow**: The robustness of our approach to initial pose perturbations reduces the need for manual adjustments during surgery, potentially streamlining the surgical workflow and reducing operating time.
- **Cross-modal Capability**: The ability to register preoperative MRI data with intraoperative optical images eliminates the need for intraoperative MRI, which is expensive, time-consuming, and not widely available.
- **Real-time Updates**: The computational efficiency of our approach enables real-time or near-real-time updates of the registration during surgery, allowing surgeons to adapt to changes in the surgical field.

These improvements could ultimately lead to better patient outcomes, reduced complications, and shorter recovery times.

8.3 Limitations and Future Directions

Despite the promising results, there are still several limitations and challenges that need to be addressed in future work:

- **Tissue Deformation**: Our current approach assumes that the brain surface geometry is relatively stable between the preoperative and intraoperative stages. Future work should explore extensions to handle tissue deformation, such as deformable NeRFs, biomechanical constraints, and incremental registration.

- **Computational Efficiency**: While our approach is computationally feasible for clinical use, there is still room for improvement in terms of rendering time and memory usage. Future work should investigate techniques to accelerate NeRF rendering and reduce memory requirements.

- **Clinical Validation**: More extensive clinical validation is needed to fully assess the potential of our approach for real-world clinical use. Future work should conduct prospective clinical studies to evaluate the impact on surgical outcomes, workflow efficiency, and surgeon satisfaction.

- **Integration with Surgical Navigation**: Integrating our approach with existing surgical navigation systems requires addressing practical considerations such as user interface design, real-time feedback, and compatibility with different hardware platforms.

In addition to addressing these limitations, future work could explore broader applications of our approach, such as other surgical domains, longitudinal registration, and multi-modal fusion.

8.4 Final Remarks

This thesis has demonstrated the potential of NeRF-based approaches for intraoperative registration, with significant improvements over traditional methods in terms of accuracy, robustness, and cross-modal capability. The combination of SSIM loss and Deep Feature Matching hypernetwork provides the best performance, achieving a mean Target Registration Error of 2.29 mm on real clinical data.

While there are still challenges to overcome, our results suggest that NeRF-based registration could become a valuable tool for image-guided neurosurgery, potentially improving surgical precision and patient outcomes. We hope that this work will inspire further research in this direction and contribute to the advancement of image-guided surgery techniques.

As medical imaging and computer vision technologies continue to evolve, we anticipate that neural implicit representations like NeRFs will play an increasingly important role in medical image analysis and surgical guidance. By combining the strengths of these representations with advanced optimization techniques and deep learning approaches, we can develop more accurate, robust, and efficient registration methods that address the challenges of intraoperative navigation and ultimately improve patient care.

Bibliography

[1] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. “Image style transfer using convolutional neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 2414–2423.

[2] Lena Maier-Hein et al. “Surgical navigation beyond visualization”. In: *Medical image analysis* 41 (2017), pp. 176–196.

[3] Ben Mildenhall et al. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis”. In: *Communications of the ACM* 65.1 (2022), pp. 99–106.

[4] Thomas Müller et al. “Instant Neural Graphics Primitives with a Multiresolution Hash Encoding”. In: *ACM Transactions on Graphics (ToG)*. Vol. 41. 4. 2022, pp. 1–15.

[5] Matthew Tancik et al. “Nerfstudio: A Modular Framework for Neural Radiance Field Development”. In: *arXiv preprint arXiv:2302.04264* (2023).

[6] Guandao Wang et al. “NeRFmm: Neural Radiance Fields from Multiview Multi-exposure Images”. In: *arXiv preprint arXiv:2010.02300* (2021).

[7] Xingtong Wang et al. “Intraoperative Registration by Cross-Modal Inverse Neural Rendering”. In: *IEEE Transactions on Medical Imaging* 40.12 (2021), pp. 3718–3729.

[8] Yunlong Wang et al. “Parallel Inversion of Neural Radiance Fields for Robust Pose Estimation”. In: *arXiv preprint arXiv:2111.12077* (2021).

[9] Zhou Wang et al. “Image quality assessment: from error visibility to structural similarity”. In: *IEEE transactions on image processing* 13.4 (2004), pp. 600–612.

[10] Lin Yen-Chen et al. “iNeRF: Inverting Neural Radiance Fields for Pose Estimation”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021.

Appendix A

Appendices

A.1 Mathematical Derivations

A.1.1 Volume Rendering in Neural Radiance Fields

The volume rendering equation used in NeRFs is given by:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt \tag{A.1}$$

where $C(\mathbf{r})$ is the color of the ray $\mathbf{r}$, $\sigma(\mathbf{r}(t))$ is the density at point $\mathbf{r}(t)$, $\mathbf{c}(\mathbf{r}(t), \mathbf{d})$ is the color at point $\mathbf{r}(t)$ when viewed from direction $\mathbf{d}$, and $T(t)$ is the accumulated transmittance:

$$T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds\right) \tag{A.2}$$

In practice, this integral is approximated using numerical quadrature:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i\delta_i))\mathbf{c}_i \tag{A.3}$$

where $\delta_i = t_{i+1} - t_i$ is the distance between adjacent samples, and T_i is the discrete accumulated transmittance:

$$T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j\delta_j\right) \tag{A.4}$$

A.1.2 Gradient of SSIM Loss with Respect to Pose Parameters

The Structural Similarity Index (SSIM) between two images x and y is defined as:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{A.5}$$

where μ_x and μ_y are the means, σ_x^2 and σ_y^2 are the variances, and σ_{xy} is the covariance of x and y. C_1 and C_2 are constants to stabilize the division.

The SSIM loss is defined as:

$$\mathcal{L}_{\text{SSIM}}(x, y) = 1 - \text{SSIM}(x, y) \tag{A.6}$$

To optimize the pose parameters θ using gradient descent, we need to compute the gradient of the SSIM loss with respect to θ:

$$\nabla_\theta \mathcal{L}_{\text{SSIM}}(x, y) = -\nabla_\theta \text{SSIM}(x, y) \tag{A.7}$$

Using the chain rule, we can express this as:

$$\nabla_\theta \text{SSIM}(x, y) = \frac{\partial \text{SSIM}(x, y)}{\partial x} \frac{\partial x}{\partial \theta} \tag{A.8}$$

where x is the rendered image, which depends on the pose parameters θ. The term $\frac{\partial x}{\partial \theta}$ involves the gradient of the volume rendering process with respect to the pose parameters, which can be computed using automatic differentiation in frameworks like PyTorch.

A.2 Implementation Details

A.2.1 NeRF Architecture

The NeRF model used in our experiments is based on the Nerfacto implementation in nerfstudio, with the following architecture:

- **Encoding**: Multiresolution hash encoding with 16 levels, each with a resolution ranging from 16 to 2048 and a feature dimension of 2.
- **MLP**: 2-layer MLP with hidden dimension 64, followed by a density head (1 output) and a 4-layer MLP with hidden dimension 64 for the color head (3 outputs).

- **Activation Functions**: ReLU for intermediate layers, exponential for density output, and sigmoid for color output.
- **Optimization**: Adam optimizer with learning rate 5×10^{-4}, trained for 30,000 iterations.

A.2.2 Hypernetwork Architecture

The hypernetwork architecture used for style transfer consists of:

- **Encoder**: Depends on the style encoding method:
  - **Y'UV**: 3-channel input, processed by a 3-layer CNN with 16, 32, and 64 filters.
  - **HOG**: HOG features extracted with cell size 8×8 and 9 orientation bins.
  - **Deep Features**: Features extracted from the conv4_2 layer of a pre-trained VGG-16 network.
- **MLP**: 3-layer MLP with hidden dimensions 256 and 512, and output dimension depending on the modulation method.
- **Modulation**: Several methods implemented:
  - **AdaIN**: Adaptive Instance Normalization, which modulates the mean and variance of features.
  - **Feature Modulation**: Direct modulation of RGB and density outputs with scale and bias parameters.

A.2.3 Loss Function Implementation

The implementation of the SSIM loss function in PyTorch:

```
def ssim_loss(x, y, window_size=11, reduction='mean'):
    """
    Structural Similarity Index loss.

    Args:
        x: First image batch (B, C, H, W)
        y: Second image batch (B, C, H, W)
        window_size: Size of the Gaussian window
        reduction: Reduction method ('mean', 'sum', or
            'none')
```

```python

    Returns:
        SSIM loss (1 - SSIM)
    """
    C1 = 0.01 ** 2
    C2 = 0.03 ** 2

    # Generate Gaussian window
    window = _create_window(window_size,
       x.shape[1]).to(x.device)

    # Calculate means
    mu1 = F.conv2d(x, window, padding=window_size//2,
       groups=x.shape[1])
    mu2 = F.conv2d(y, window, padding=window_size//2,
       groups=y.shape[1])

    mu1_sq = mu1.pow(2)
    mu2_sq = mu2.pow(2)
    mu1_mu2 = mu1 * mu2

    # Calculate variances and covariance
    sigma1_sq = F.conv2d(x * x, window,
       padding=window_size//2, groups=x.shape[1]) -
       mu1_sq
    sigma2_sq = F.conv2d(y * y, window,
       padding=window_size//2, groups=y.shape[1]) -
       mu2_sq
    sigma12 = F.conv2d(x * y, window,
       padding=window_size//2, groups=x.shape[1]) -
       mu1_mu2

    # Calculate SSIM
    ssim_map = ((2 * mu1_mu2 + C1) * (2 * sigma12 +
       C2)) / ((mu1_sq + mu2_sq + C1) * (sigma1_sq +
       sigma2_sq + C2))

    # Convert to loss (1 - SSIM)
    loss = 1 - ssim_map

    if reduction == 'mean':
        return loss.mean()
    elif reduction == 'sum':
        return loss.sum()
    else:  # 'none'
        return loss
```

A.3 Additional Results

A.3.1 Detailed Performance Metrics

Table A.1 shows detailed performance metrics for the best-performing configuration (SSIM loss + Deep Feature Matching hypernetwork) across different perturbation levels.

Table A.1: Detailed performance metrics for SSIM + Deep Feature Matching

Perturbation Level	**Trans. Error (mm)**	**Rot. Error (deg)**	**SSIM**
Small (1-5 mm, 1-5 deg)	0.87 ± 0.42	1.23 ± 0.61	0.94 ± 0.03
Medium (5-10 mm, 5-10 deg)	1.32 ± 0.65	1.89 ± 0.94	0.91 ± 0.04
Large (10-20 mm, 10-20 deg)	2.18 ± 1.09	3.12 ± 1.56	0.87 ± 0.06
Very Large (20-30 mm, 20-30 deg)	3.76 ± 1.88	5.41 ± 2.71	0.79 ± 0.09

A.3.2 Convergence Analysis

Figure A.1 shows the convergence behavior of different optimization algorithms (Gradient Descent, Adam, and L-BFGS) for the SSIM loss function.

Figure A.1: Convergence behavior of different optimization algorithms for SSIM loss

The results show that L-BFGS converges the fastest, followed by Adam and then Gradient Descent. However, L-BFGS has higher memory requirements and can be less stable for some cases.

A.3.3 Ablation Studies

Table A.2 shows the results of ablation studies to evaluate the contribution of different components of our framework.

The ablation studies confirm that both the SSIM loss and the Deep Feature Matching hypernetwork contribute significantly to the performance of

Table A.2: Ablation studies for different components of the framework

Configuration	Trans. Error (mm)	Rot. Error (deg
Full Framework (SSIM + Deep Feature Matching)	1.35 ± 0.71	2.18 ± 1.15
Without Hypernetwork	2.43 ± 1.21	3.67 ± 1.89
With L2 Loss instead of SSIM	2.29 ± 1.14	3.42 ± 1.71
Without Multi-resolution Sampling	1.87 ± 0.94	2.95 ± 1.52
Without Pose Regularization	1.62 ± 0.83	2.41 ± 1.27

our framework. The multi-resolution sampling and pose regularization also provide modest improvements.

A.4 Code Samples

A.4.1 Registration Pipeline

The main registration pipeline:

```python
def register_image(nerf_model, target_image, initial_pose,
                   loss_fn, hypernetwork=None,
                   num_iterations=500, lr=0.01):
    """
    Register a target image with a NeRF model.

    Args:
        nerf_model: Pre-trained NeRF model
        target_image: Target intraoperative image
        initial_pose: Initial camera pose estimate
        loss_fn: Loss function to use
        hypernetwork: Optional hypernetwork for style transfer
        num_iterations: Maximum number of iterations
        lr: Learning rate

    Returns:
        Optimized camera pose
        Loss history
        Rendered image at optimized pose
    """
    # Initialize pose parameters
    pose_params = torch.tensor(initial_pose, requires_grad=True)

    # Setup optimizer
    optimizer = torch.optim.Adam([pose_params], lr=lr)

    # Initialize loss history
    loss_history = []

    # Optimization loop
    for i in range(num_iterations):
        # Zero gradients
        optimizer.zero_grad()

        # Render image from current pose
        rendered_image = nerf_model.render(pose_params)

        # Apply hypernetwork if provided
        if hypernetwork is not None:
            style_code = hypernetwork.encode(target_image)
            style_params = hypernetwork(style_code)
            rendered_image = hypernetwork.apply_style(rendered_image, style_params)

        # Compute loss
        loss = loss_fn(rendered_image, target_image)

        # Backpropagate and update pose
        loss.backward()
        optimizer.step()

        # Record loss
        loss_history.append(loss.item())

        # Check convergence
        if i > 50 and abs(loss_history[-1] - loss_history[-50]) < 1e-5:
            break

    # Render final image
    with torch.no_grad():
        final_image = nerf_model.render(pose_params)
        if hypernetwork is not None:
            style_code = hypernetwork.encode(target_image)
            style_params = hypernetwork(style_code)
            final_image = hypernetwork.apply_style(final_image, style_params)

    return pose_params.detach(), loss_history, final_image
```

A.4.2 Hypernetwork Implementation

Implementation of the Deep Feature Matching hypernetwork:

```python
class DeepFeatureMatchingHypernetwork(nn.Module):
    """
    Hypernetwork for style transfer using deep feature matching.
    """
    def __init__(self, feature_extractor, output_dim):
        super().__init__()
        self.feature_extractor = feature_extractor

        # Freeze feature extractor
        for param in self.feature_extractor.parameters():
            param.requires_grad = False

        # Feature dimension from the feature extractor
        self.feature_dim = 512  # For VGG16 conv4_2

        # MLP to generate style parameters
        self.mlp = nn.Sequential(
            nn.Linear(self.feature_dim, 256),
            nn.ReLU(),
            nn.Linear(256, 512),
            nn.ReLU(),
            nn.Linear(512, output_dim)
        )

    def encode(self, image):
        """Extract features from the image."""
        # Preprocess image if needed
        if image.shape[1] == 3:  # RGB
            # VGG expects normalized images
            mean = torch.tensor([0.485, 0.456, 0.406]).view(1, 3, 1, 1).to(image.device)
            std = torch.tensor([0.229, 0.224, 0.225]).view(1, 3, 1, 1).to(image.device)
            image = (image - mean) / std

        # Extract features
        features = self.feature_extractor(image)

        # Global average pooling
        features = F.adaptive_avg_pool2d(features, (1, 1)).squeeze(-1).squeeze(-1)

        return features

    def forward(self, style_code):
        """Generate style parameters from style code."""
        return self.mlp(style_code)

    def apply_style(self, image, style_params):
        """Apply style parameters to the image."""
        # Split style parameters into scale and bias for RGB
        rgb_scale, rgb_bias = style_params.chunk(2, dim=1)

        # Reshape for broadcasting
        rgb_scale = rgb_scale.view(-1, 3, 1, 1)
        rgb_bias = rgb_bias.view(-1, 3, 1, 1)

        # Apply modulation
        styled_image = image * rgb_scale + rgb_bias

        # Ensure values are in valid range
        styled_image = torch.clamp(styled_image, 0, 1)

        return styled_image
```

A.5 Dataset Details

A.5.1 Synthetic Dataset

The synthetic dataset was generated using a pre-trained NeRF model of a brain surface, with the following specifications:

- **Number of Images**: 100 target images
- **Resolution**: 256 $\times$ 256 pixels
- **Camera Parameters**:
  - Focal Length: 35 mm
  - Field of View: 60 degrees
  - Distance from Brain Surface: 100-150 mm
- **Perturbation Levels**:
  - Small: 1-5 mm translation, 1-5 degrees rotation
  - Medium: 5-10 mm translation, 5-10 degrees rotation
  - Large: 10-20 mm translation, 10-20 degrees rotation
  - Very Large: 20-30 mm translation, 20-30 degrees rotation

A.5.2 Real Dataset

The real dataset consists of 20 cases from neurosurgical procedures, with the following characteristics:

- **Preoperative MRI**:
  - T1-weighted with gadolinium contrast
  - 1 mm isotropic resolution
  - Acquired 1-7 days before surgery
- **Intraoperative Images**:
  - Optical images captured with a surgical microscope
  - Resolution: 1920 $\times$ 1080 pixels
  - Captured after dura opening and before tumor resection
- **Ground Truth**:

- Manually annotated correspondences between MRI and intraoperative images
- 10-15 landmarks per case, marked by experienced neurosurgeons
- Estimated accuracy of manual annotations: 1-2 mm

Due to privacy concerns, the real dataset cannot be publicly shared, but anonymized samples and statistical summaries are provided for reference.