**CHAPTER 5: SYSTEM DESIGN**

**5.1 System Design & Methodology**

The image simulation system is designed to take a GeoTIFF image as input and simulate how a camera would capture it in an unknown environment. The methodology follows a structured pipeline consisting of image loading, parameter configuration, marker placement, and simulation output.

The key components of the system design are:

* **Graphical User Interface (GUI):** Built using PyQt and PyQtGraph, providing an interactive environment for parameter adjustments and image visualization.
* **Camera Parameters Handling:** Users can set parameters such as detector height, width, FOV, focal length, and altitude to define the camera specifications.
* **Marker-Based Simulation:** Users can place markers using latitude-longitude coordinates to define the simulation area. Single-point mode allows one marker, while multi-point mode enables continuous simulation between two points.
* **Footprint Preview:** The system provides a footprint visualization before actual simulation to help users understand the expected output coverage.
* **Image Processing Engine:** The core simulation logic processes the input GeoTIFF based on selected camera parameters, generating the simulated image.

This structured approach ensures that the system effectively meets the objective of simulating camera behavior for testing and design purposes.

**5.2 Data Structure Design / Process Design / Structure Design**

**Data Structure Design**

The system primarily works with the following data structures:

* **Camera Parameter Dictionary:** Stores values such as detector size, FOV, focal length, and altitude.
* **GeoTIFF Image Data:** Handled as multi-dimensional arrays for processing.
* **Marker Data:** Stores latitude-longitude pairs for single and multi-point modes.

**Process Design**

The image simulation follows these steps:

1. **Load Image:** User selects a GeoTIFF image.
2. **Set Camera Parameters:** Parameters are either manually entered or selected from the preloaded camera types via a combo box.
3. **Mark Position(s):** User marks a single or two-point location using the mouse or by entering coordinates.
4. **Preview Footprint:** The system calculates and displays the expected coverage.
5. **Simulate Image:** The system processes the image according to camera parameters and outputs the simulated image.
6. **Save or Clear:** Users can save results or reset parameters and markers.

**5.3 Input / Output and Interface Design**

**5.3.1 State Transition Diagram (Optional)**

If required, a state transition diagram can be included to depict the various states of the system, such as **Idle → Image Loaded → Parameters Set → Simulation Performed → Output Displayed**.

**5.3.2 Samples of Forms, Reports, and Interface**

* **Forms & Inputs:** The GUI provides **QLineEdit fields** for numerical input and a **QComboBox** for selecting preloaded camera types.
* **Interactive Controls:** **Buttons** for loading, saving, and clearing data, along with **QDials** for adjusting roll, pitch, yaw, and tilt angle.
* **Visualization:** The **PyQtGraph canvas** displays the input and output images with an interactive histogram.

**5.3.3 Access Control / Security (If Applicable)**

Since this is a local simulation tool, access control is minimal. However, basic safeguards such as **input validation** and **error handling** are implemented to prevent incorrect data entry.

**CHAPTER 6: IMPLEMENTATION**

**6.1 Implementation Platform / Environment**

The image simulation system is developed using Python and implemented on a Windows/Linux environment. The key components of the implementation platform are:

* **Programming Language:** Python
* **GUI Framework:** PyQt5 (for creating an interactive graphical interface)
* **Image Processing Library:** GDAL (for handling GeoTIFF images)
* **Visualization Library:** PyQtGraph (for displaying images and histograms)
* **Mathematical Computation:** NumPy (for handling array-based image transformations)
* **Development Environment:** PyCharm / VS Code / Jupyter Notebook
* **Hardware Requirements:** A system with a minimum of **8GB RAM, multi-core CPU, and GPU acceleration (optional for enhanced processing)**

The system runs efficiently on standard computing hardware, with optimizations to handle large GeoTIFF files.

**6.2 Process / Program / Technology / Modules Specification(s)**

**Process Flow**

1. **Loading the Input Image**
   * Users load a **GeoTIFF** file using a file selection dialog.
   * The image data is read using the **GDAL library** and processed into an array for simulation.
2. **Setting Camera Parameters**
   * Users manually enter or select a predefined **camera type** from a **QComboBox**.
   * Parameters such as **detector size, focal length, field of view, and altitude** are applied.
3. **Marking Positions (Single or Multi-Point Mode)**
   * Users **click on the image or enter latitude-longitude coordinates** to set markers.
   * The system converts these to pixel coordinates for processing.
4. **Footprint Visualization**
   * A **footprint button** calculates and overlays the expected simulated area on the original image.
5. **Simulating the Image**
   * The system applies **geometric transformations** based on camera parameters.
   * The modified image is displayed on the **PyQtGraph canvas**.
6. **Saving / Resetting Data**
   * Users can save results, reset inputs, or clear markers as needed.

**Technologies & Modules Used**

* **PyQt5:** GUI framework for interactive controls.
* **PyQtGraph:** Efficient image rendering and histogram display.
* **GDAL:** Reads and processes GeoTIFF images.
* **NumPy:** Handles mathematical transformations for image simulation.

**6.3 Findings / Results / Outcomes**

The implementation of the system resulted in the following outcomes:

* **Accurate Image Simulation:** The system successfully simulates how a camera would capture an image in an unknown environment.
* **Real-Time Parameter Adjustment:** Users can modify camera parameters and observe changes dynamically.
* **Footprint Visualization:** The footprint feature helps users understand how the final image will appear before simulation.
* **Multi-Mode Simulation:** Both **single-point and multi-point** simulation modes function as expected.
* **Efficient Processing:** The use of **NumPy and GDAL** ensures that large GeoTIFF images are processed efficiently.

**6.4 Result Analysis / Comparison / Deliberations**

The system was tested with various input images and camera parameters to evaluate its performance. The key observations include:

* **Comparison with Theoretical Expectations:** The simulated images align with expected outputs based on camera equations.
* **Performance Metrics:** Processing time varies based on image resolution and selected parameters. Higher-resolution images take longer to process.
* **User Interaction:** The GUI provides an intuitive interface for parameter adjustments, making the system user-friendly.
* **Challenges:** Some minor challenges were faced in marker precision, which were mitigated using coordinate transformations.

Overall, the system meets its objectives of providing a **reliable and efficient** image simulation tool for camera design and testing.

**CHAPTER 7: TESTING**

**7.1 Testing Plan / Strategy**

The testing phase ensures that the image simulation system functions correctly and meets the intended objectives. The strategy involves:

**Testing Types Used:**

1. **Unit Testing** – Individual components (e.g., image loading, parameter inputs, footprint preview) are tested separately.
2. **Integration Testing** – Ensures that different modules (e.g., GUI, parameter selection, simulation engine) work together correctly.
3. **Functional Testing** – Verifies that all features (e.g., single/multi-point simulation, marker placement) perform as expected.
4. **Performance Testing** – Checks system efficiency with different image sizes and parameter variations.
5. **User Acceptance Testing (UAT)** – Ensures the system meets user requirements for camera testing and design.

**Testing Environment:**

* **Platform:** Windows/Linux
* **Software Dependencies:** Python, PyQt5, PyQtGraph, GDAL, NumPy
* **Hardware:** System with at least 8GB RAM and multi-core processor

**7.2 Test Results and Analysis**

Multiple test cases were executed to verify the system’s performance and accuracy. The following table summarizes key test cases:

**7.2.1 Test Cases**

| **Test ID** | **Test Condition** | **Expected Output** | **Actual Output** | **Remarks** |
| --- | --- | --- | --- | --- |
| TC-01 | Load GeoTIFF image | Image loads successfully | Image loads correctly | Pass |
| TC-02 | Enter valid camera parameters | Parameters accepted and applied | Parameters stored and used | Pass |
| TC-03 | Enter invalid camera parameters | Error message displayed | Error handled correctly | Pass |
| TC-04 | Single-point marker placement | Marker appears at correct lat-long | Marker placed accurately | Pass |
| TC-05 | Multi-point marker placement | Two markers placed correctly | Markers appear as expected | Pass |
| TC-06 | Footprint preview | Correct footprint displayed | Footprint aligns with expected region | Pass |
| TC-07 | Run image simulation | Simulated image is generated | Image output is accurate | Pass |
| TC-08 | Save simulated image | Image saved in directory | File saved successfully | Pass |
| TC-09 | Clear parameters | All fields reset | Fields cleared properly | Pass |
| TC-10 | Performance with high-res image | Processing within reasonable time | Acceptable performance | Pass (minor delay) |

**Analysis of Test Results:**

* **Functionality:** All core features performed as expected, with accurate image simulation.
* **Usability:** The GUI is intuitive, with smooth user interactions.
* **Performance:** The system handles various image resolutions well, but higher-resolution images slightly increase processing time.
* **Error Handling:** Proper validation prevents invalid inputs, improving system robustness.

Overall, the testing confirms that the system is **stable, functional, and reliable** for its intended purpose.

This completes **Chapter 7: Testing**. Let me know when you need the next section!

Here’s the revised **Chapter 8: Conclusion and Summary, Limitations** as per your request:

**CHAPTER 8: CONCLUSION AND SUMMARY, LIMITATIONS**

**8.1 Conclusion and Summary**

The image simulation system was developed to evaluate how a camera would capture images in an unknown environment. By incorporating camera-specific parameters such as detector height, width, field of view (FOV), focal length, and altitude, the system provides an effective way to analyze and design cameras before actual deployment.

The system meets its objectives by providing a **useful tool for camera testing and design**, helping researchers and engineers analyze camera performance in different conditions.

**8.2 Limitations**

**Processing Time for High-Resolution Images:**

* When working with very large GeoTIFF images, processing can take longer than expected. Optimizations in computation can help reduce delays.

**Limited Camera Parameter Customization:**

* The system currently supports predefined parameters and manual input. Future versions could allow users to import real-world camera calibration data.

**No Real-Time 3D Visualization:**

* The system operates in a 2D simulation environment. Adding a **3D rendering option** could improve visualization and understanding of camera perspectives.

**Georeferencing Accuracy:**

* While the system accurately places markers based on latitude-longitude coordinates, minor discrepancies may arise due to projection transformations.

**No External Data Integration:**

* Currently, the system processes only user-provided images. Future enhancements could support integration with external satellite data sources.

Despite its limitations, the system serves as a **valuable tool for testing and designing camera configurations**.