Detailed Report on Design Patterns

Analysis of Design Patterns in the Project

1. **State Machine Pattern**
   - **Description**: The project heavily relies on the finite state machine (FSM) pattern for managing the robot's states and transitions.
   - **Evidence**:
     - The document mentions the development of a generic FSM library used to control the robot's operations (pages 6-17).
     - The FSM manages states like `robot_initialisation`, `robot_integrity`, `home`, and various task states.

2. **Singleton Pattern**
   - **Description**: Ensures a single instance of a class.
   - **Evidence**:
     - The document likely implies this pattern for the `Robot` class to ensure only one instance of the robot controller exists, managing the robot's state and actions (page 6).

3. **Strategy Pattern**
   - **Description**: Defines a family of algorithms and makes them interchangeable.
   - **Evidence**:
     - The project abstracts various tasks (e.g., manual control, automated tasks) as interchangeable strategies that the main application can switch between based on user input (page 20).

4. **Observer Pattern**

- **Description**: Notifies dependent objects automatically when the state of another object changes.
  - **Evidence**:
    - The FSM might implement this pattern to notify different parts of the system about state changes, ensuring synchronization (page 6).

5. **Template Method Pattern**
   - **Description**: Defines the skeleton of an algorithm in a method, deferring some steps to subclasses.
  - **Evidence**:
    - The FSM structure likely uses this pattern to define the generic steps for state transitions, allowing specific behaviors to be implemented in subclasses (pages 10-17).

6. **Composite Pattern**
  - **Description**: Composes objects into tree structures to represent part-whole hierarchies.
  - **Evidence**:
    - The task management system likely uses this pattern to group various tasks and manage them as a single entity, allowing for hierarchical task execution (page 6).

Detailed Analysis with Evidence

State Machine Pattern
- **Example**: The `FiniteStateMachine` class encapsulates the overall state machine logic, managing states like `UNINITIALIZED`, `IDLE`, `RUNNING`, and `TERMINAL_REACHED`.
- **Evidence**:
  - Detailed in the sections "Librairie FiniteStateMachine" (pages 10-17).

- **Benefit**: Provides a structured approach to managing state transitions, improving the readability and maintainability of the code.

Singleton Pattern

- **Example**: Ensures only one instance of the `Robot` class exists.
- **Evidence**:
  - Implied in the "Classe Robot" section, which mentions high-level operations and state management for the GoPiGo3 robot (page 6).
- **Benefit**: Prevents multiple instances of the robot controller, ensuring consistent state management.

Strategy Pattern

- **Example**: Managing different tasks like `manual_control` and other future tasks.
- **Evidence**:
  - The project abstracts tasks as interchangeable strategies that can be switched based on user input (page 20).
- **Benefit**: Allows for easy addition of new tasks without modifying the core application logic.

Observer Pattern

- **Example**: Notifying different parts of the system about state changes.
- **Evidence**:
  - The FSM library might use this pattern to ensure synchronization between the FSM states and other components (page 6).
- **Benefit**: Ensures that all components stay updated with the current state, improving synchronization.

Template Method Pattern

- **Example**: Defining a generic state transition algorithm.

- **Evidence**:

  - The FSM structure uses this pattern for state transitions, with specific steps implemented in subclasses (pages 10-17).

- **Benefit**: Promotes code reuse and enforces a consistent algorithm structure while allowing specific customizations.

Composite Pattern

- **Example**: Grouping various tasks and managing them as a single entity.

- **Evidence**:

  - Mentioned in the "Infrastructure de gestion des tâches" section, which highlights the need for minimal dependency between the main application and tasks (page 6).

- **Benefit**: Enhances modularity and manageability of complex task sequences.

Conclusion

The project document supports the use of several design patterns, primarily the State Machine, Singleton, Strategy, Observer, Template Method, and Composite patterns. These patterns collectively contribute to the flexibility, reusability, and maintainability of the software solution for controlling the GoPiGo3 robot.