PDW (Pulse Descriptor Word) Simulator

A Python-based simulator for modeling radar-sensor interactions and generating Pulse Descriptor Words (PDWs). This simulator models radar emissions and sensor detections with configurable parameters and error models.

Project Structure

```
pdw-simulator/
│
├── src/
│   └── pdw_simulator/
│       ├── __init__.py
│       ├── main.py               # Core simulation execution
│       ├── models.py             # Core classes (Scenario, Radar, 
Sensor)
│       ├── radar_properties.py   # Radar-specific functions
│       ├── sensor_properties.py  # Sensor-specific functions
│       └── scenario_geometry_functions.py  # Geometry calculations
│
├── tests/
│   ├── __init__.py
│   ├── conftest.py              # Test fixtures and configuration
│   ├── test_models.py           # Tests for core classes
│   ├── test_integration.py      # Integration tests
│   ├── test_radar_properties.py # Tests for radar functions
│   ├── test_sensor_properties.py # Tests for sensor functions
│   └── fixtures/
│       ├── __init__.py
│       └── test_config.yaml     # Test configuration data
│
├── apps/
│   └── app.py                   # Streamlit web interface
│
├── docs/
│   ├── README.md                # Package documentation
│   ├── API.md                   # API documentation
│   └── examples/
│       └── basic_simulation.md  # Usage examples
│
├── examples/
│   └── basic_simulation.py      # Example scripts
│
├── .gitignore
├── LICENSE
├── README.md
├── pyproject.toml
├── pytest.ini
```

PROF

```
├── requirements.txt
└── setup.py
```

Core Components

1. Scenario Management (`models.py`)

- Scenario class: Controls simulation environment and time progression
- Radar class: Models radar behavior and properties
- Sensor class: Models sensor detection and measurement capabilities

2. Radar Properties (`radar_properties.py`)

- Pulse generation patterns (Fixed, Stagger, Switched, Jitter)
- Frequency management
- Pulse width control
- Antenna lobe patterns
- Rotation patterns

3. Sensor Properties (`sensor_properties.py`)

- Detection probability models
- Measurement error models
- Parameter measurements (TOA, Frequency, Amplitude, etc.)

4. Geometry Calculations (`scenario_geometry_functions.py`)

- Position and trajectory calculations
- Unit management using Pint

Setup Instructions

PROF

1. Environment Setup

```
# Create and activate a virtual environment
python -m venv venv
source venv/bin/activate  # On Windows, use: venv\Scripts\activate

# Clone the repository
git clone https://your-repository-url/pdw-simulator.git
cd pdw-simulator
```

2. Installation

```
# Install in development mode
pip install -e .
```

```
# Install required dependencies
pip install -r requirements.txt
```

3. Configuration

Create a `dataconfig.yaml` file in your working directory with your simulation parameters. Example structure:

```
scenario:
  start_time: 0
  end_time: 10
  time_step: 0.1

radars:
  - name: "Radar1"
    start_position: [0, 0]
    rotation_type: "constant"
    rotation_params:
      t0: 0
      alpha0: 0
      T_rot: 4
    pri_type: "fixed"
    pri_params:
      pri: 0.001
    # ... additional radar parameters ...

sensors:
  - name: "Sensor1"
    start_position: [1000, 1000]
    # ... sensor parameters ...
```

PROF

Running the Simulator

Basic Usage

```
# From the project root directory
python -m src.pdw_simulator.main
```

This will:

1. Read configuration from `dataconfig.yaml`
2. Run the simulation
3. Generate output in `pdw_output_dataconfig.csv`

Output Format

The simulator generates a CSV file with the following columns:

- Time: Simulation time
- SensorID: Identifier of the detecting sensor
- RadarID: Identifier of the detected radar
- TOA: Time of Arrival
- Amplitude: Signal amplitude
- Frequency: Signal frequency
- PulseWidth: Pulse width
- AOA: Angle of Arrival

Error Models

The simulator supports various error models for sensor measurements:

- Constant
- Linear
- Sinusoidal
- Gaussian
- Uniform

Configure these in your `dataconfig.yaml` file under sensor configurations.

Dependencies

- numpy: Numerical computations
- pint: Unit handling
- pyyaml: Configuration file parsing
- jax: Additional numerical computations
- scipy: Scientific computations

Contributing

1. Fork the repository
2. Create a feature branch
3. Make your changes
4. Submit a pull request

Testing

```
# Run tests
pytest tests/
```

License

Support

Acknowledgments