EH-OWRF for IoT - User's Manual

1 Introduction

Library for simulating a hybrid Optical Wireless VLC(downlink)/RF(uplink) indoor WSN for IoT with Energy Harvesting (EH) capabilities for indoor spaces. The communication model includes simulation of PHY and MAC (Unslotted CSMA-CA) layers. VLC employes ON-OFF Keying modulation taking into account both LOS & Diffuse power contributions, where the RF uplink is based on 802.15.4 @2.4GHz. Duty-cycling is also assumed and the period can be set accordingly. For the EH, the incident optical power to the PV panels is calculated and the five-parameters PV panel model is used to predict the maximum possible harvested power. It is also possible to include the effect of sunlight for both communication and EH purposes.

Installation - Step by Step instructions

1. Open the terminal on your system.
2. Create a virtual environment :
   - **On Windows:**

     ```
     python -m venv hybridEH
     ```

   - **On macOS/Linux:**

     ```
     python3 -m venv hybridEH
     ```

3. Activate the virtual environment:
   - **On Windows:**

     ```
     .\hybridEH\Scripts\activate
     ```

   - **On macOS/Linux:**

     ```
     source hybridEH/bin/activate
     ```

   The library depends on the following libraries" **Numpy, Matplotlib, Scipy, Pyswarms**. You can install the tested versions of the libraries manually executing this command in the terminal:

   ```
   pip install numpy==1.26.4 matplotlib==3.8.0 scipy==1.11.4 pyswarms==1.3.0
   ```

4. Next, navigate to the directory you want to clone the project:

   ```
   cd path/to/your/project
   ```

5. Clone the repository executing the following command on your terminal:

   ```
   git clone https://github.com/AAslan94/Hybrid_EH_IoT
   ```

6. Navigate inside the project:

   ```
   cd "Hybrid_EH_IoT"
   ```

7. You can now execute the default scenario:

```
python3 run.py
```

or edit the files on your preferred source code editor.

2 Start-up guide

Design.py

To run your scenario, you need to define the parameters by modifying the *design.py* file.

- **room_x**: The dimensions of the room in meters.
- **refl_x**: The reflectivity values (0,1) of the walls, ceiling floor.
- **amb_Lx**: The dimensions of the ambient light source (window) in meters.
- **r_sensor**: A Numpy array representing the position of the SNs. For *N* SNs, the array should be of size (N,3).
- **FOV_sensor**: The Field of View of the SNs' photodiode in radians.
- **nR_sensor**: A Numpy array representing the normal to the SNs photodiode vectors, i.e. the orientation of the receiver.
- **A_pd**: The active area of the photodiode in m^2.
- **r_master**: A Numpy array representing the positions of the Master Nodes (Base Station).
- **m_master_vlc**: The Lambertian order of the VLC transmitters.
- **nS_master_vlc**: A Numpy array representing the normal to the VLC transmitters vectors, i.e. their orientation.
- **PT_master**: The optical power transmitted from each VLC transmitter in Watts.
- **Sensitivity**: The sensitivity of the RF receivers in dBm.
- **Rb_master**: The downlink effective bit-rate in bits/second.
- **Rb_sensor**: The uplink effective bit-rate in bits/second.
- **bits_master**: The downlink packet length in bits
- **bits_sensor**: The uplink packet length in bits
- **no_bounces**: Number of bounces considered for diffuse lighting simulation.
- **IWU**: The current consumed by the SN in the Wake-Up phase in Amperes.
- **tWU**: The time it takes for the SN to wake up in seconds.
- **Iadc**: The current consumed by the ADC peripheral of the SNs MCU in Amperes.
- **Iact**: The current consumed by the MCU when active in Amperes.
- **Vmcu**: The operating voltage of the MCU in Volts.
- **Vsensor**: The operating voltage of the sensors in Volts.
- **Isensor**: A Numpy array representing the current consumed by the sensors in Amperes.
- **Itia**: The current consumed in Amperes by the Trans-Impendance amplifier $\approx$ the quiscent current of the Op-Amp.
- **Icca**: The current consumed in Amperes when performing CCA (should include the current consumed by the MCU CPU).

- **Tcycle**: The duty cycle in seconds.
- **m_leds**: Lambertian order of the Lighting LEDs.
- **r_leds**: A Numpy array representing the position of the Lighting LEDs.
- **nS_leds**: A Numpy array representing the normal to the Lighting LEDs vectors, i.e. their orientation.
- **PT_leds**: The optical power transmitted from each Lighting LED in Watts.
- **nR_solar**: A Numpy array representing the normal to the PV panels vectors, i.e. orientation (**see next section**).
- **A_solar**: Active area of the PV panels in m^2.
- **Isc**: The short-circuit current of the PV panel in Amperes under STC.
- **Voc**: The open-circuit voltage of the PV panel in Volts under STC.
- **Imp**: Current at the maximum power point in Amperes under STC.
- **Vmp**: Voltage at the maximum power point in Volts under STC.
- **N**: Number of PV cells, connected in series, which comprise the PV panel.
- **A**: First approximation of the ideality factor of the diode in the PV panel equivalent 5-parameters circuit.

linear_power_current.py

The current consumption of the SNs when RF transmitting depends on the required output RF power. To have an estimate for your transceiver you can find from the datasheet of your desired RF transmitter the minimum and maximum possible RF output power in dB and their corresponding current consumption in mA and replace these values in the linear_power_current.py script. Then you can run the script:

```
python3 linear_power_current.py
```

The script assumes a linear relationship between output power (dB) and current (mA):

$$current = a * power + b \tag{1}$$

You can replace the values of a,b in the rf.py file:

```
def min_tx_power_level(power):
     return power,a* power + b
```

h_comm.py

This file contains the logic for simulating the hybrid VLC/RF network. The network is represented by an object from the *HybridWSN* class. You need to simply create a HybridWSN object as:

```
network = HybridWSN()
```

The parameters for your design will be fetched from the *design.py* file. The results are stored as attributes of the object. With the following, you can access the energy consumed by each SN each cycle in Joules.

```
network.energy_cycle
```

To see the energy consumed from each SN every day in Joules:

```
network.energy_day
```

With the following, you can see the total (LOS + diffuse) SNR in dB and the total BER for the downlink.

```
network.vlc_snr_total
network.vlc_snr_los
network.vlc_snr_diffuse
network.ber_downlink
```

opt_solar_orientation.py

Before running the Energy Harvesting simulation (next section), it is strongly advised to execute this script by typing in the terminal:

```
python3 opt_solar_orientation.py
```

Based on the parameters of your scenario, the notebook uses the PySwarms (Particle Swarm Optimization) library to calculate the near-optimal orientation of the PV panels for maximum energy harvesting. The results by default will be written in the *"op_l_sun.npy"* file and will be loaded to the *nR_solar* variable. In case you do not want to take into account the ambient sunlight, run *opt_solar_orientation_nsun.py* instead and in *design.py* file:

```
op = np.load('op_l_nsun.npy')
```

nrg_harvesting.py

The simulation assumes a *Energy Harvesting* subsystem consisting of a PV panel, a MPPT circuit (SPV1050) and a battery. The logic for the energy harvesting - related simulations are incorporated in the *EH* class. To create a new object:

```
eh = EH()
```

Once again, the related parameters will be fetched from the *desing.py* file. All results take into account both scenarios: when sunlight streams through the window and when it does not. The simulated efficiency of the PV panels can be accessed then by:

```
eh.eff_sun
eh.eff_no_sun
```

The maximum power in Watts coming from the PV panel:

```
eh.p_out_sun
eh.p_out_no_sun
```

The maximum power out of the PV panel, accounting for the losses in the MPPT-enabled buck-boost converter:

```
eh.pext_sun
eh.pext_no_sun
```

The current to the battery, considering the charging voltage of the battery (Vbatt) and the fact that SPV1050 can deliver up to 70mA current.

```
eh.iext_sun
eh.iext_no_sun
```

The total energy harvested per hour for both scenarios:

```
eh.esun_h
eh.ensun_h
```

You can use the objects *network* and *eh* to approximate the PV panel size needed for your application by:

```
calculate_min_solar_panel_area(wsn = network,nrg = eh,h1 =3,h2 =8,m = 0.1)
```

where h1 is the average time in hours with sunlight per day, h2 is the average time in hours with LED lighting per day and m is a safety margin, i.e 0.1 = 10%.

If you are using another Energy harvesting MPPT chip, you can set the efficiency of your converter by altering the values in the *buck_boost_eff()* method and setting the maximum current in the *cap_current()* method.

If you just want to approximate the battery life of your network (in days) without any energy harvesting:

```
battery_life_est(mAh = 250,voltage = 3.7,ws = network,i_self = 1e-6)
```

where *mAh* is the battery life in mAh, voltage is the nominal voltage of the battery, ws is the *HybridWSN* object and *i_self* is the self-discharge current of the battery.

3 Default Scenario

You can execute the *"run.py"* example for a quick start. This scenario assumes:

- A 10mx10mx3m room is assumed. A 2mx2m window is placed with it center at (x=0,y=5m,height=1.5m).
- There are 25 ceiling-mounted LEDs uniformly distributed in a grid pattern across the area from (x, y) = (1m, 1m) to (x, y) = (9m, 9m). Among these, only the lamp positioned at the center of the ceiling (x = 5m, y = 5m, height = 3m) serves as a VLC transmitter for communication purposes, while the remaining 24 lamps are dedicated solely to lighting.
- 49 Sensor Nodes are uniformly distributed in a grid pattern across the floor spanning from (x,y) = (0.3m,0.3m) to (x,y) = (9.7m,9.7m).
- The Sensor Nodes are equipped with the a PV panel modeled based on the Voltaic P121 [3] model.

You can run the same scenario without ambient sunlight using the "run_no_sun.py" script.

4 Other

- The MAC layer algorithm will be run 2000 times by default and the average values are used to estimate the power consumption of the nodes for each cycle. You can adjust this by changing the number when the function is called inside the *HybridWSN* class. Every time a HybridWSN object is created, the MAC algorithm executes and the results are saved in the "time_data.npz" file. To save time, if you want to try different configurations keeping the number of nodes constant, you can comment out line 66 of the code and uncomment line 67 (h_comm.py).
- Modelling of the PV panels is based on the five-parameters equivalent circuit. By default, the algorithm by Stormelli et al [1] is used. The method by Brano et al [2], is also implemented. For using it, you must alter the following in the *"nrg_harvesting.py"* file.

```
D = SolarPanel(..., method = 'Brano')
```

5 How it works?

Documentation of the internal workings of the library will follow soon.

6 References

1 V. Stornelli, M. Muttillo, T. de Rubeis, and I. Nardi, "A new simplified five-parameter estimation method for single-diode model of photovoltaic panels," Energies, vol. 12, no. 22, 2019.

2 V. Lo Brano, A. Orioli, C. Giuseppina, and A. Di Gangi, "An improved five-parameter model for photovoltaic modules," Solar Energy Materials and Solar Cells, vol. 94, pp. 1358–1370, 08 2010.

3 https://voltaicsystems.com/content/P121_ds.pdf