Hunger-Driven Chemotaxis

Implementation Plan

1 Overview

The experiment extends the existing phototaxis setup with an internal **battery** that depletes over time and recharges when the robot is near the light source. A new **interoceptive HomeoUnit** monitors battery level, acting as a "hunger drive" that influences motor behaviour through the homeostat's own homeostatic dynamics.

2 Architecture Mapping

Concept	Implementation
Robot with sensors/motors	Already exists: `KheperaRobot` + `HOMEO_DiffMotor` / `HOMEO_LightSensor` transducers
Battery that discharges	**New**: `Battery` object in `KheperaSimulator`, tracked as a body/state in the simulation
Interoceptive "hunger" sensor	**New**: `HOMEO_BatterySensor` transducer + a `HomeoUnitInput` reading battery level
Battery recharges near light	**New**: logic in `advanceSim()` that recharges battery proportionally to light irradiance at robot position

3 Step-by-step Plan

3.1 Step 1: Add Battery to KheperaSimulator

Add a simple battery model to `KheperaSimulator.py`:

```
class KheperaBattery:
    def __init__(self, capacity=1.0, discharge_rate=0.001,
                 recharge_factor=0.01):
        self.capacity = capacity
        self.level = capacity          # starts full
        self.discharge_rate = discharge_rate
        self.recharge_factor = recharge_factor

    def tick(self, irradiance=0.0):
        """Discharge, then recharge proportionally to
        irradiance at robot's position."""
        self.level -= self.discharge_rate
        self.level += self.recharge_factor * irradiance
        self.level = max(0.0, min(self.capacity, self.level))
        return self.level

    def range(self):
```

```
        return (0.0, self.capacity)
```

3.2 Step 2: Wire battery into KheperaRobot and advanceSim

- Give `KheperaRobot` an optional `battery` attribute (default `None` for backward compatibility).
- In `KheperaSimulation.advanceSim()`, if the robot has a battery, compute the total irradiance at the robot's current position from all detectable lights, then call `battery.tick(irradiance)`.

The battery discharges every simulation step and recharges only when the robot is close enough to the light to receive non-negligible irradiance—exactly the "hunger" dynamic from the research note.

3.3 Step 3: Add HOMEO_BatterySensor transducer

In `RobotSimulator/Transducer.py`, following the existing `HOMEO_LightSensor` pattern:

```
class HOMEO_BatterySensor(Transducer):
    """Interoceptive transducer that reads the
    robot's battery level."""

    def __init__(self, robotRef):
        self.robot = robotRef

    def read(self):
        return self.robot.battery.level

    def range(self):
        return self.robot.battery.range()

    def act(self):
        raise TransducerException(
            "Battery sensor cannot act")
```

This bridges the robot's internal state (battery) to the homeostat via a standard `HomeoUnitInput`, just like `HOMEO_LightSensor` bridges external light readings.

3.4 Step 4: Create world setup function with battery

In `KheperaSimulator.py`, add a new world setup method:

```
def kheperaBraitenberg2_HOMEO_World_battery(self):
    """Like kheperaBraitenberg2_HOMEO_World
    but with a battery."""
    world = self.kheperaBraitenberg2_HOMEO_World()
    self.allBodies[self.robotName].battery = \
        KheperaBattery()
    return world
```

3.5 Step 5: Create experiment function in HomeoExperiments.py

Add a new GA experiment function that builds an **8-unit homeostat**:

Unit	Type	Role
Left Motor	`HomeoUnitNewtonianActuator`	Drives left wheel
Right Motor	`HomeoUnitNewtonianActuator`	Drives right wheel
Left Eye	`HomeoUnitNewtonian`	Processes left eye input
Right Eye	`HomeoUnitNewtonian`	Processes right eye input
Hunger	`HomeoUnitNewtonian`	Processes battery level
Left Sensor	`HomeoUnitInput`	Reads left eye irradiance
Right Sensor	`HomeoUnitInput`	Reads right eye irradiance
Battery Sensor	`HomeoUnitInput`	Reads battery level

The **Hunger unit** is a standard `HomeoUnitNewtonian` that receives input from the Battery Sensor (and potentially from other units). Its essential parameter values are evolved by the GA. When battery is low, the Battery Sensor feeds a high "hunger" deviation into the Hunger unit, which propagates through the homeostat's connections to influence motor outputs—the homeostatic dynamics themselves determine *how* hunger affects behaviour.

Genome structure: 5 evolved units $\times$ 4 essential params + 5 evolved units $\times$ 8 total units = **60 genes** (up from 40 in the standard phototaxis experiment).

3.6 Step 6: Define fitness function

Three options:

Option A — Battery survival
Fitness = number of ticks the battery remains above 0 (or above some threshold).

Option B — Average battery level
Robots that stay near the light maintain high battery over the run.

Option C — Combined
Final distance to target $\times$ battery penalty.

A new method on the backend (e.g., `averageBatteryLevel()`) would report the battery metric.

3.7 Step 7: Lesion studies (future)

Once a successful controller is evolved, systematic lesion studies would remove the Hunger unit or lesion specific connections (e.g., Hunger→Motor) and compare performance. These don't require new code—just experiment variants that disable specific connections after loading an evolved genome.

4 Key Design Decisions

1. **How many units?** 8-unit homeostat with distinct interoceptive channel, or feed battery into existing eye units (losing the separate "interoceptive" channel)?
2. **Fitness measure**: Battery survival time vs. average battery level vs. distance-based?
3. **Battery parameters**: Discharge rate and recharge factor determine task difficulty. Fixed or included in the GA?
4. **Genome size**: 60 genes (from 40) is manageable but increases search space—may need larger population or more generations.

5 Files to Modify/Create

File	Change
`KheperaSimulator.py`	Add `KheperaBattery` class; wire into `KheperaRobot` and `advanceSim`; add battery world setup
`Transducer.py`	Add `HOMEO_BatterySensor` class
`HomeoExperiments.py`	Add hunger-phototaxis experiment function
`HomeoGenAlgGui.py`	Add experiment to combo box; possibly add battery-based fitness path
`SimulatorBackend.py`	Add `averageBatteryLevel()` or similar method
`HomeoGATest.py`	Tests for new experiment attributes