Szilard's Engine Simulation

Term Project, PHYSICS 3P03

Jay Mehta

400285158

Submitted on March 31, 2022

Department of Physics & Astronomy, McMaster University

About the Szilard's Engine model

In 1929, Leo Szilard sought to simplify Maxwell's original conception with an imaginary device that became known as Szilard's engine (Szilard, 1929). In this paper, I attempt to demonstrate a model of this thought experiment, which uses one bit of information about the position of a thermalized molecule in a box to extract $\mathbf{W = k_B T \ln 2}$ of work. The second law of thermodynamics remains valid because, according to Landauer's principle, erasure of the information dissipates the same amount of heat at the minimum. Here, I present a numerical simulation of a Maxwell demon similar to the Szilard engine, that consisted of a single particle of ideal gas molecule confined within a box of finite length and "weakly coupled" to a thermal bath at temperature T. A demon ("an intelligent being") determines the location of the particle (and store this information in its memory register) and inserts a partition in the middle of this box that can slide in such a way as to extract useful work from the reversible isothermal expansion of the single-particle ideal gas, without any change in the free energy of the system.

In case of quasi-static process, the particle can be in equilibrium with the environment outside the box at temperature T, and integrating the work gained over the volume expansion of V /2, we obtain,

$$W = \int_{V/2}^{V} P\, dV = \int_{V/2}^{V} \frac{k_B T}{V}\, dV = k_B T \ln 2 \tag{1}$$

Going through the Code

In this project, essentially, I am going to exhibit an illustration of a concrete relationship between Thermodynamic Entropy and the Information Theory, indicating that information is indeed physical, and Szilard's engine is a simplified thought experiment that shows how we can extract work from an engine by simply possessing certain piece of information about it.

In particular, I have considered 2 different possibilities (either the position is 'known' or 'unknown') in 3 separate cases as follows, where the user may specify the input from the set {'left', 'right', 'any'}:

case = 'left': When we know that the particle can be found at a random location but on the left side of the partition wall.

case = 'right': When we know that the particle can be found at a random location but on the right side of the partition wall.

case = 'any': When we do not know the position of the particle, so it may be found randomly in either side of the partition wall with the probability $\frac{1}{2}$ in each side of the box.

Conventionally, I have assumed all the corresponding constants to be equal to one. Boltzmann's Constant: $k_B = 1$, Heat Bath Temperature: $T_{bath} = 1$. Also the value of the work done by the molecule in increasing the volume is taken positive if the wall moves towards the right end of the box and negative otherwise. This enables us to demonstrate that in the 'any' case, the total work is, in principle, equal to zero. Please refer to the appendices where I have attached an example of a single instance of my main function of the simulation as a numpy array, that essentially replicates the behaviour of the particle's motion as it moves within the box, once.

Data Analysis and Results

In the figure 1, we are looking at the case where the particle position has been known to the demon and hence it extracts work from the system. The value approaches the theoretical value we obtained in the equation 1, as the accuracy of the numerical integration method increases.

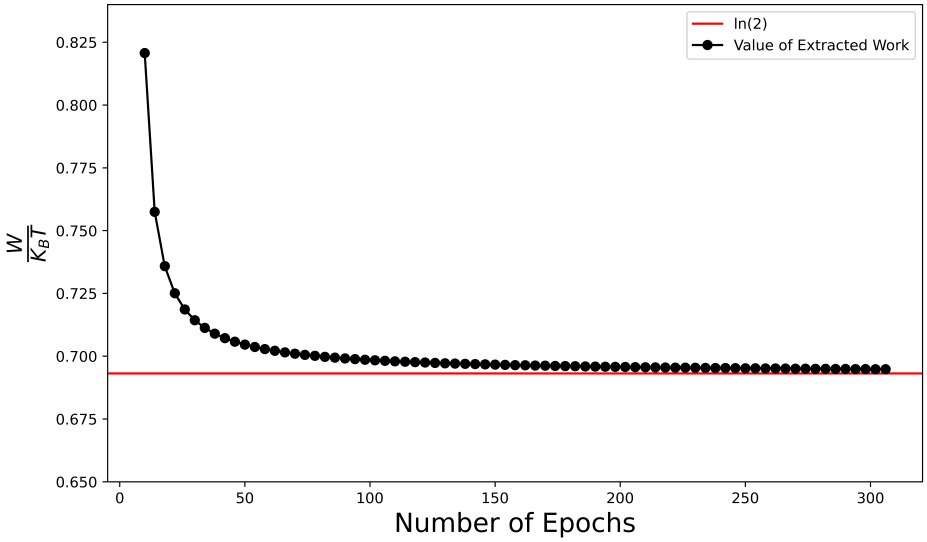

Figure 1: W/k_BT value approaches to ln(2) asymptotically as we increase the dimensions of the box after each epoch, consequently increasing the precision in numerically evaluating the inverse volume integral.

Each data point in the figure 2 corresponds to the average value of the work obtained from the heat bath (at T) after executing the simulation many times (1000, in this case). And the similar runs have been performed 60 times in the mentioned figure.

From this, it becomes evident enough to conclude that upon knowing the information of the particle's position, the demon changes the entropy of the system (box + demon). Otherwise, the average stays fairly around zero.

Also, the generated data files (.csv) containing these plotted values have been attached

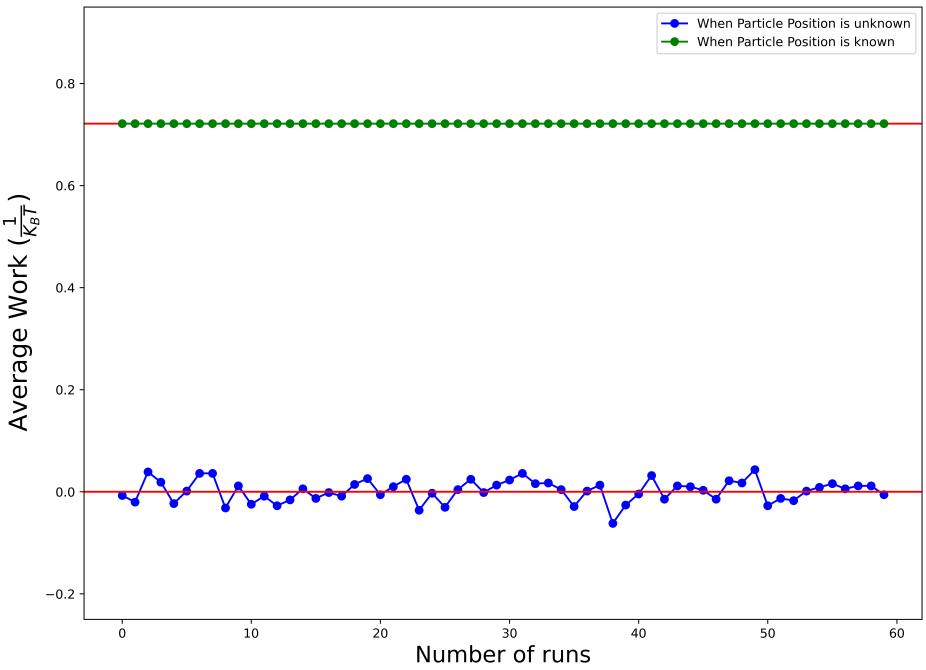

Figure 2: Average value of the work obtained from the reservoir. The points coloured in green portray the scenario when the position of the particle is 'known' and the points coloured in blue portray the scenario when the position of the particle is 'unknown'.

alongside.

Conclusion

The most succinct manifestation of the second law of thermodynamics is the limitation imposed by the Landauer's principle on the amount of heat a Maxwell demon can convert into free energy per single bit of information obtained in a measurement (Koski et al., 2014).

In the ideal limit, the amount of work extracted turns out to be equal to, $\mathbf{W = k_B T \ln 2}$, where k_B is the Boltzmann's constant. As a result, the entropy of the gas decreases by an amount $\mathbf{\Delta S = k_B \ln 2}$, corresponding to a Shannon Entropy of one bit (Shannon, 1951), defined as the reduction in uncertainty on inspecting the outcome of a process with probability 1/2, e.g., flipping an unbiased coin. The Szilard engine has, thus, converted information about the particle's location into useful energy, showing that information can serve as a type of fuel.

Therefore, the results of this experimental data certainly agree with our expectations and the justification of those results are concisely explained in this summary.

References

Szilard, L. (1929). Über die entropieverminderung in einem thermodynamischen system bei eingriffen intelligenter wesen. *Zeitschrift für Physik*, *53*(11), 840–856.

Shannon, C. E. (1951). Prediction and entropy of printed english. *Bell system technical journal*, *30*(1), 50–64.

Koski, J. V., Maisi, V. F., Pekola, J. P., & Averin, D. V. (2014). Experimental realization of a szilard engine with a single electron. *Proceedings of the National Academy of Sciences*, *111*(38), 13786–13789. https://doi.org/10.1073/pnas.1406966111

Appendix

Following is an instance of my main function of the simulation model, displayed as a numpy array. Call the function "display()" in the code, or uncomment certain lines in the python file to visualize this.

```
[[1 1 1 1 1 1 1 1 1 1 9 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 9 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 9 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 9 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 9 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 9 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 9 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 9 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 9 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 9 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 9 1 1 1 1 1 1 0 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 9 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 9 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 9 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 9 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 9 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 9 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 9 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 9 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 9 1 1 1 1 1 1 1 1 1 1]]
```

Figure 3: Input parameters: size of the box = 20×20, *column of* $9s$ = *wall*, 0 = *particle*, $1s$ = *empty space. Computationally speaking, putting* $1s$ *to represent the empty volume of the box is a clever way to determine the current value of the volume during each instance of the run that can be further used to evaluate the intergral.*

The above implemented initialization of the 'box' has been done using the following block of code:

```python
import numpy as np
import math
import matplotlib.pyplot as plt

# initializes the box with r rows, c columns and partition wall
def initialize(wall, r=20, c=21):
  box = []
  for i in range(r):
    temp = []
    for j in range(c):
      # 1 represents void space within the box walls
      temp.append((1))
    box.append(temp)
  box = np.array(box)
  # a column of 9s represents the wall
  box[:, wall] = 9
```

```
  return box

# generates a random position of the particle
def particleGenerator(wall, r=20, c=21, case="right"):
  try:
    column1 = np.random.randint(wall+1, c)
  except:
    column1 = np.random.randint(0, wall)
  try:
    column2 = np.random.randint(0, wall)
  except:
    column2 = np.random.randint(wall+1, c)
  if case == "right":
    column = column1
  elif case == "left":
    column = column2
  elif case == "any":
    temp = np.random.randint(0,2)
    if temp == 0:
      column = column1
    else:
      column = column2
  row = np.random.randint(0, r)
  return (row, column)
```

The complete python code, that I have designed to run the emulated model in various possible situations, can be found in the attached submission files (.ipynb). User may define the number of rows & columns to determine the dimensions of the box and the case they would like to evaluate.