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


# Plotting the output control variable (High (-))
def output_controller_high_negative(x):
    if -0.09 >= x >= -0.15:
        return_value = 1
    elif -0.03 >= x >= -0.09:
        return_value = -(100 / 6) * x - 0.5
    else:
        return_value = 0
    return return_value


# Plotting the output control variable (Small (-))
def output_controller_small_negative(x):
    if 0 >= x >= -0.03:
        return_value = -(100 / 3) * x
    elif -0.03 >= x >= -0.06:
        return_value = (100 / 3) * x + 2
    else:
        return_value = 0
    return return_value


# Plotting the output control variable (Good)
def output_controller_good(x):
    if 0 >= x >= -0.03:
        return_value = (100 / 3) * x + 1
    elif 0.03 >= x >= 0:
        return_value = - (100 / 3) * x + 1
    else:
        return_value = 0
    return return_value


# Plotting the output control variable (Small (+))
def output_controller_small_positive(x):
    if 0.06 >= x >= 0.03:
        return_value = - (100 / 3) * x + 2
    elif 0.03 >= x >= 0:
        return_value = (100 / 3) * x
    else:
        return_value = 0
    return return_value


# Plotting the output control variable (High (+))
def output_controller_high_positive(x):
    if 0.15 >= x >= 0.09:
        return_value = 1
    elif 0.09 >= x >= 0.03:
        return_value = (100 / 6) * x - 0.5
    else:
        return_value = 0
    return return_value


# Define x-axis for the output control value
t = np.linspace(-0.15, 0.15, 400)
counter = 0
high_negative = np.zeros(shape=t.shape)
small_negative = np.zeros(shape=t.shape)
good = np.zeros(shape=t.shape)
small_positive = np.zeros(shape=t.shape)
high_positive = np.zeros(shape=t.shape)

for i in t:
    high_negative[counter] = output_controller_high_negative(i)
    small_negative[counter] = output_controller_small_negative(i)
    good[counter] = output_controller_good(i)
    small_positive[counter] = output_controller_small_positive(i)
    high_positive[counter] = output_controller_high_positive(i)

    counter += 1

```

```python
plt.ylabel("Membership")
plt.xlabel("$\delta$")
plt.title('Fuzzy Partition for the control variable')
plt.plot(t, high_negative, label='High Infection rate (-)')
plt.plot(t, small_negative, label='Low Infection rate (-)')
plt.plot(t, good, label='Good Infection rate')
plt.plot(t, small_positive, label='Low Infection rate (+)')
plt.plot(t, high_positive, label='High Infection rate (+)')

plt.legend(bbox_to_anchor=(0.98, 1), loc="upper left")

plt.show()


# Calculating the membership for the infected bots rate (High)
def controller_high(x):
    if x >= 0.8:
        return 1
    elif x <= 0.6:
        return 0
    else:
        value = 5 * x - 3
        return value


# Calculating the membership for the infected bots rate (Good)
def controller_good(x):
    if 0.65 >= x >= 0.6:
        return_value = -20 * x + 13
    elif 0.6 >= x >= 0.55:
        return_value = (20 * x) - 11
    else:
        return_value = 0
    return return_value


# Calculating the membership for the infected bots rate (Low)
def controller_small(x):
    if x <= 0.4:
        return 1
    elif x >= 0.6:
        return 0
    else:
        value = -5 * x + 3
        return value


# Define x-axis for the infected bots rate
k = np.linspace(-0.05, 1, 400)

counter = 0
good_triangle = np.zeros(shape=k.shape)
high_triangle = np.zeros(shape=k.shape)
small_triangle = np.zeros(shape=k.shape)
for x in k:
    good_triangle[counter] = controller_good(x)
    high_triangle[counter] = controller_high(x)
    small_triangle[counter] = controller_small(x)
    counter += 1

plt.ylabel("Membership")
plt.xlabel("$\pi$")
plt.title('Fuzzy partition of the current percentage of the infected bots')
plt.plot(k, small_triangle, label='Low Infection rate')
plt.plot(k, good_triangle, label='Good Infection rate')
plt.plot(k, high_triangle, label='High Infection rate')
plt.xticks(np.arange(0, 1.1, 0.1))
plt.legend(bbox_to_anchor=(0.98, 1), loc="upper left")

plt.show()


# Calculating the membership for the effective infection rate(High)
def effective_controller_high(x):
    if 1 >= x >= 0.2:
        return 1
    elif 0 <= x <= 0.2:
        value = 5 * x
        return value
    else:
        return 0


# Calculating the membership for the effective infection rate(Good)
def effective_controller_good(x):
    if 0 >= x >= -0.2:
        return_value = 5 * x + 1
    elif 0.2 >= x >= 0:
        return_value = -5 * x + 1
    else:
        return_value = 0
    return return_value


# Calculating the membership for the effective infection rate(Small)
def effective_controller_small(x):
    if -0.2 >= x >= -1:
        return 1
    elif 0 >= x >= -0.2:
        value = -5 * x
        return value
    else:
        return 0

# Define x-axis for the current effective infection rate
m = np.linspace(-1, 1, 1000)

counter = 0
good_triangle = np.zeros(shape=m.shape)
high_triangle = np.zeros(shape=m.shape)
small_triangle = np.zeros(shape=m.shape)

for x in m:
    good_triangle[counter] = effective_controller_good(x)
    high_triangle[counter] = effective_controller_high(x)
    small_triangle[counter] = effective_controller_small(x)
    counter += 1

plt.ylabel("Membership")
plt.xlabel("$\dot\pi$")
plt.title('Fuzzy partition of the current effective infection')
plt.plot(m, small_triangle, label='Low Infection rate')
plt.plot(m, good_triangle, label='Good Infection rate')
plt.plot(m, high_triangle, label='High Infection rate')
plt.xticks(np.arange(-1, 1.1, 0.2))
plt.legend(bbox_to_anchor=(0.98, 1), loc="upper left")

plt.show()

```