Vehicle 1 Lab

AGNEY PRASEED

We aim to simulate the behaviour of Vehicle 1, which is equipped with a single sensor that has a direct relationship with the stimulus source

We initialise and define the pygame setting variables, which are used for the window dimensions, fps, color and font size using the following code

```
“””
pygame.init()


WIDTH, HEIGHT = 1200, 600

screen = pygame.display.set_mode((WIDTH, HEIGHT))

pygame.display.set_caption("Braitenberg Vehicle 1 Simulation")


pygame.font.init()

font = pygame.font.SysFont("Arial", 24)


clock = pygame.time.Clock()

fps = 60


WHITE = (255, 255, 255)

YELLOW = (255, 255, 0)

RED = (255, 0, 0)

GREEN = (0, 255, 0)
```

“””

Next we define the Source (the Sun) as a Circle class, it has a position, radius and a color as parameters. We define a function draw that will render our sun on a given surface.

Source parameters :

“”

```
def __init__(self, position, radius=30, color=RED):
        self.position = pygame.math.Vector2(position)
        self.radius   = radius
        self.color    = color
```

“””

The position parameter is a vector.

The draw function :

“”

```
def draw(self, surface):
        pygame.draw.circle(surface, self.color, self.position, self.radius)
```

“”

Next we define the Vehicle class and the sensor class. We will draw the main vehicle body and its light sensor as two concentric circles.

We have the following parameters for Vehicle class position, direction, radius, color, speed_scalling

“”

```
self.position = pygame.math.Vector2(position)
self.direction = direction
```

self.radius = radius

self.color = color

self.speed_scalling = 50

“”

Now the draw function in the Vehicle class, we first draw the main body of the vehicle 1 using

“

pygame.draw.circle(surface, self.color, self.position, self.radius)

“

Now we define the sensor, we are using a circle on top on our vehicle to define the sensor.

For the sensor we define the arguments

Sensor_radius, sensor_offset, sensor_position and sensor_color

The offset of the sensor will be the distance from the vehicle’s center to the center of the sensor circle so

“”

self.sensor_offset = self.radius + self.sensor_radius

“”

For the position of our sensor, we initialise it as a vector that points straight upward by “sensor_offset”.

In Pygame, the coordinate system has its origin (0, 0) in the top-left corner, with positive x going right and positive y going down. Hence we need to add a minus to the magnitude.

Our vector is :

“”

pygame.math.Vector2(0, -self.sensor_offset)

“”

Next we rotate the vector in the direction of our vehicle using
“

.rotate(self.direction) function
“

The sensor is always in front of our vehicle, we add our vector to the vehicle’s position. This gives us the correct absolute (x,y) for our sensor.

Hence

“”

self.sensor_position = self.position +

pygame.math.Vector2(0, -self.sensor_offset).rotate(self.direction)

“”

We define a function called “calculate_sensor_position” to calculate the distance between Sun and our sensor.

The distance_to() in pygame will calculate the Euclidean distance between two vectors.

Next we define the move function :

We set the direction as a vector with magnitude 1, pointing in the upward direction and then rotated by self.direction.

For our Vehicle 1, the speed and distance are inversely propotional.

Speed = scalling parameter * (1 / distance from the sun)

We then update our position with the magnitude of speed in the straight path.

“

```
self.position += direction_vector * speed
```

”

After moving the vehicle, we calculate the new sensor position and re attach it to our vehicle.

To debug we can use the following code in pygame

“””

```
text = font.render(
          f"Distance to sun: {distance:.2f} \n speed : {speed}", True, WHITE)
      screen.blit(text, (10, 10))
```

“””

Now we define our Setup, the source and vehicle

“”

```
sun     = Circle((600, 300), radius=30, color=YELLOW)
vehicle = Vehicle((300, 500), 45)
```

“”

The sun is placed in the middle of our screen and vechicle at (300, 500) inclined at angle 45.

Setup for running our code (Main Loop) :

“””

```
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    screen.fill((0, 0, 0))        # Clear screen to black
    sun.draw(screen)              # Draw the sun
    vehicle.move(sun.position)
    vehicle.draw(screen)          # Draw the vehicle + sensor
    pygame.display.flip()         # Present the new frame
    clock.tick(fps)               # Enforce 60 FPS

pygame.quit()
```

“””

The call to “”screen.fill((0, 0, 0))”” at the start of each frame serves to clear out the previous frame’s contents by repainting the entire window black. Pygame’s display surface is persistent: once you draw something, it stays until you explicitly erase or overwrite it.

Pygame draws every frame into an off-screen buffer (the “back buffer”) rather than directly to the screen. “”pygame.display.flip()”” swaps or copies whatever is currently in the back buffer to the visible front buffer.