1 Introdução

O objetivo desse laboratório foi implementar o algorítimo *Particle Swarm Otimization* (PSO) no código pré-dispnibilizado pelo professor. Para isso, foi necessário implementar diversos métodos, os quais serão explicados nas próximas seções.

2 Código

O código das classes *Particle* e *ParticleSwarmOptimization* está disponibilizado abaixo em pseudo-código.

```
class Particle:


    def __init__(self, lower_bound, upper_bound):

        self.x=vetor de zeros
        self.v=vetor de neros
        self.best=vetor de zeros
        self.best_value=-inf



class ParticleSwarmOptimization:

    def __init__(self, hyperparams, lower_bound, upper_bound):

        self.lower_bound = lower_bound
        self.upper_bound = upper_bound
        self.delta=upper_bound-lower_bound


        self.w=hyperparams.inertia_weight
        self.phip=hyperparams.cognitive_parameter
        self.phig=hyperparams.social_parameter
        self.num_particles=hyperparams.num_particles


        self.i=0
        self.particles=[Particles]


        self.best_position=vetor de zeros
        self.best_value=-inf


    def get_best_position(self):

        return self.best_position


    def get_best_value(self):

        return self.best_value

    def get_position_to_evaluate(self):
```

```
        rp=random.uniform(0.0,1.0)
        rg=random.uniform(0.0,1.0)

        self.particles[self.i].v = self.w * self.particles[self.i].v + self.phip * rp * (
            self.particles[self.i].best - self.particles[self.i].x) + self.phig * rg * (
                self.best_position - self.particles[self.i].x)

        self.particles[self.i].v=min(max(self.particles[self.i].v,-self.delta),self.delta)

        self.particles[self.i].x=self.particles[self.i].x+self.particles[self.i].v

        self.particles[self.i].x = min(max(self.lower_bound, self.particles[self.i].x),
            self.upper_bound)


        return self.particles[self.i].x

    def advance_generation(self):

        self.i+=1
        se self.i>= self.num_particles:
            self.i=0


    def notify_evaluation(self, value):


        se value > self.best_value:
            self.best_position=self.particles[self.i].x
            self.best_value=value

        se value > self.particles[self.i].best_value:
            self.particles[self.i].best=self.particles[self.i].x
            self.particles[self.i].best_value=value


        self.advance_generation()
```

3 Resultados do teste

O objetivo do teste era obter o valor do máximo da função $q(x) = -[(x-1)^2 + (x-2)^2 + (x-3)^2]$. O máximo dessa função é trivial e vale $\mathbf{x} = [1\ 2\ 3]$. O programa supracitado foi testado com 40 partículas, para 1000 iterações cada. Os resultados de convergência estão citados nas figuras 1, 2, 3.

4 Resultado do seguimento de linha

Foi implementado ainda o método *evaluate*, o qual deveria, a cada iteração, retornar a recompensa para o percurso feito pelo carrinho. O pseudo-código está mostrado abaixo (com elementos já previamente implementados):

```
def evaluate (self):
    linear, angular = self.line_follower.get_velocity()  # v_k, omega_k
    error, detection = self.line_follower.line_sensor.get_error()  # e_k
    track_tangent = self.track.get_tangent(self.line_follower.pose.position)  # t_k
    robot_direction = Vector2(cos(self.line_follower.pose.rotation),
        sin(self.line_follower.pose.rotation))  # r_k
    dot_product = track_tangent.dot(robot_direction)  # dot(r_k, t_k)
    se detection is false:
        error=1
```

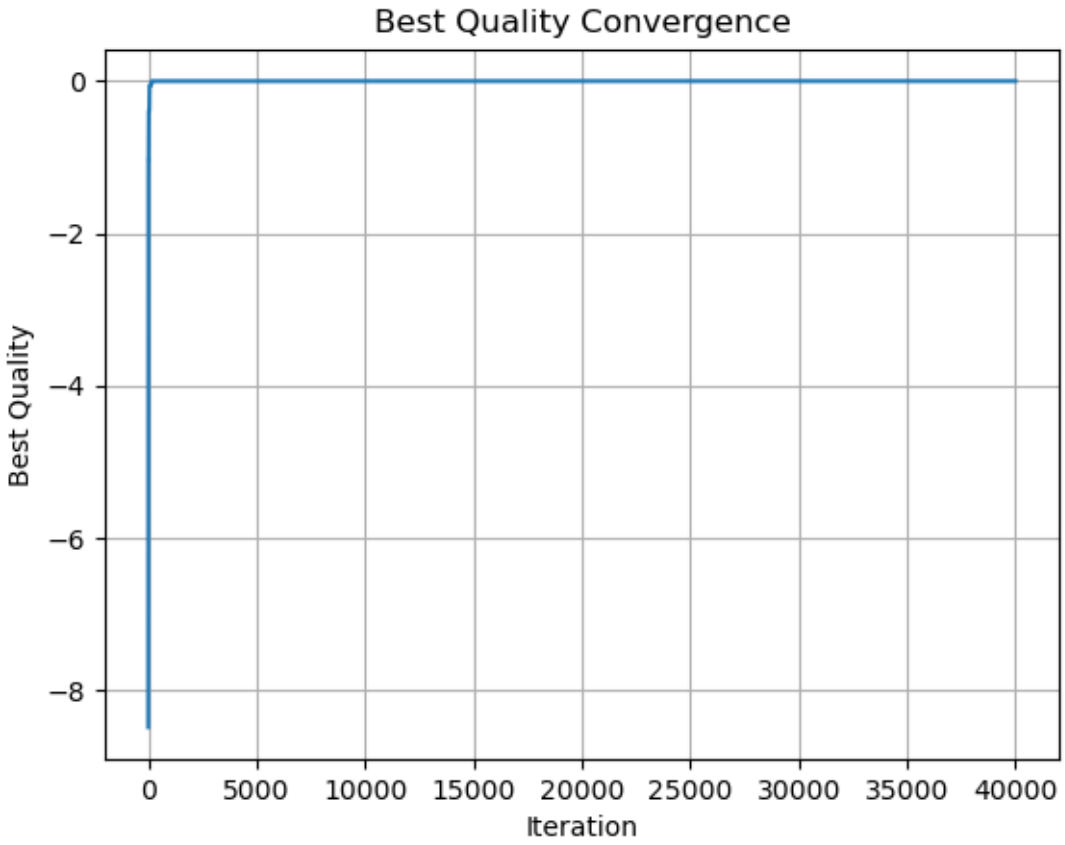

Figura 1: Resultado do teste com a função q

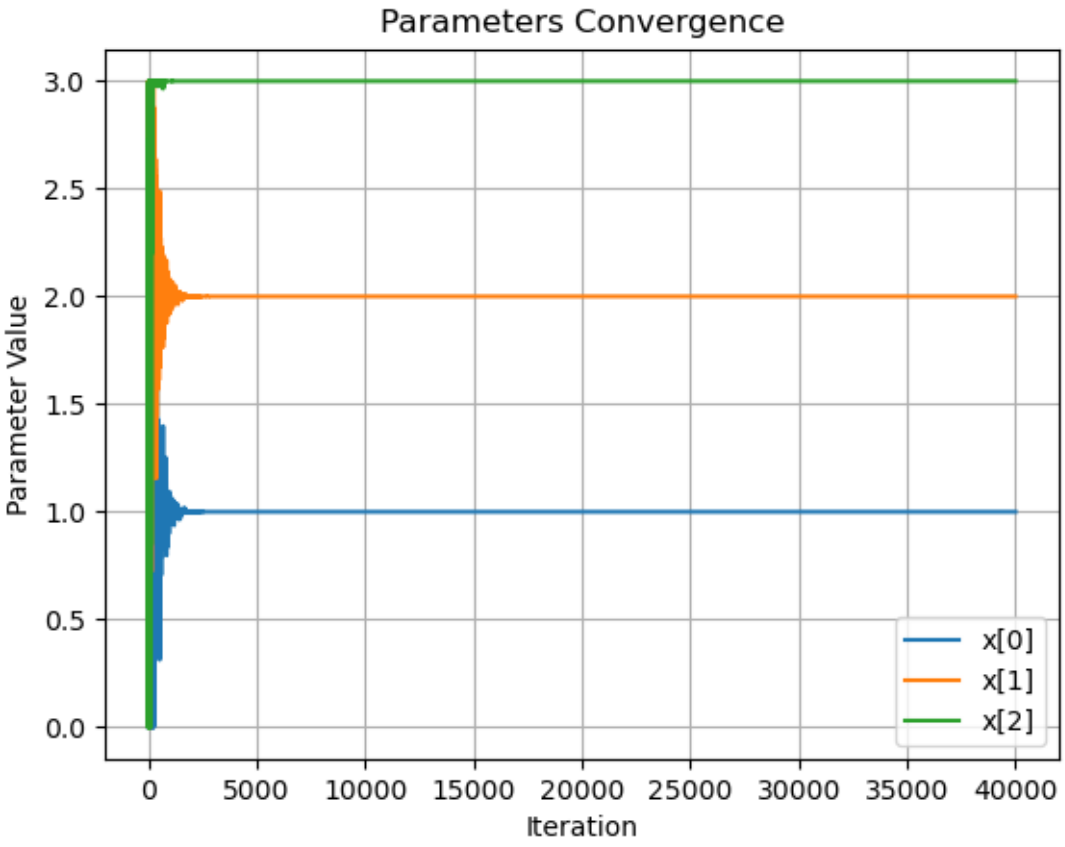

Figura 2: Resultado de convergência dos parâmetros

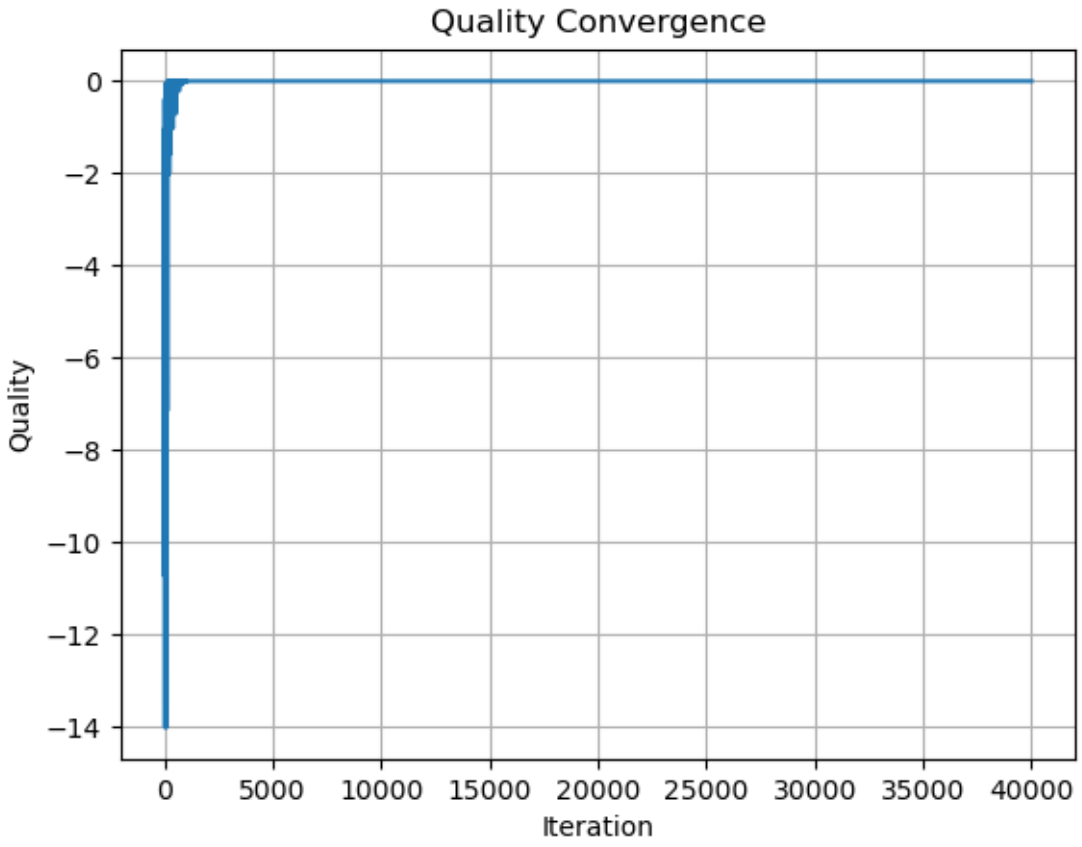

Figura 3: Resultado de convergência da função q

```
    reward=linear*dot_product-0.5*fabs(error)

    return reward
```

Note que o erro escolhido para caso a linha não fosse detectada foi 1. Tal valor foi escolhido após diversos testes, sendo o valor que gerou melhores resultados (vide a figura 4), convergindo para um trajeto razoável após aproximadamente 3000 iterações de treinamento. Além disso, as figuras 5, 6 e 7 mostram resultados de convergência dos outros parâmetros.

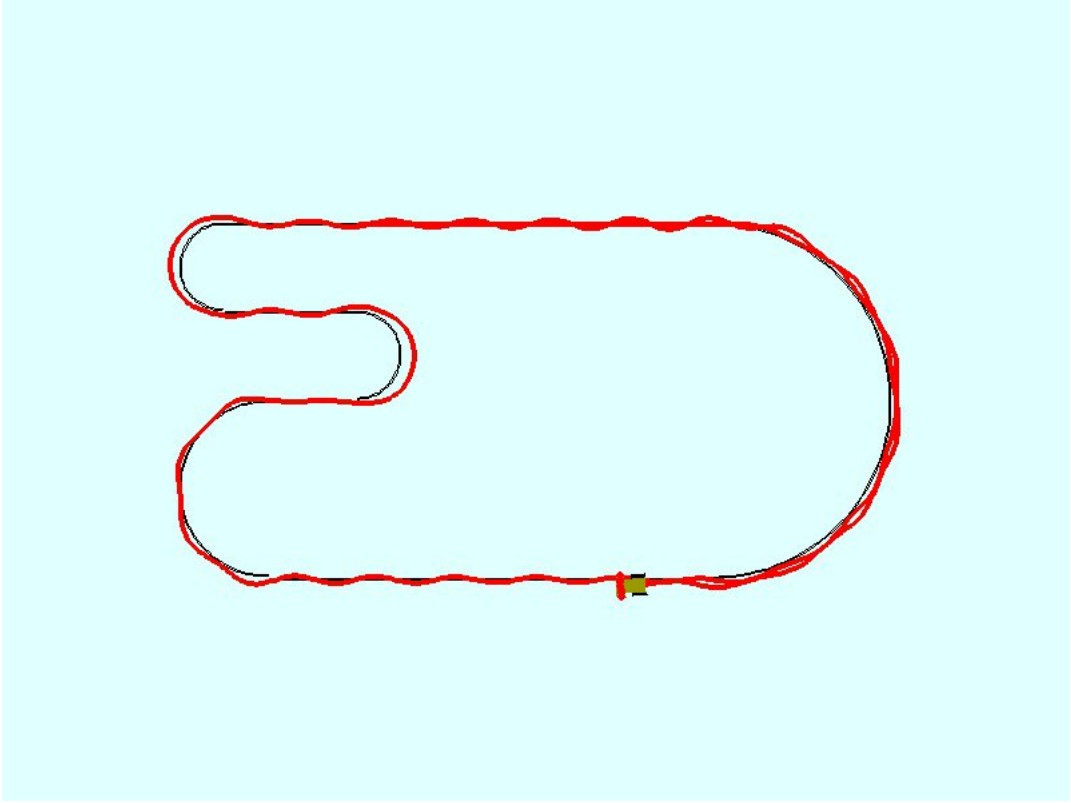

Figura 4: Resultado do carrinho seguidor de linha após cerca de 3000 iterações

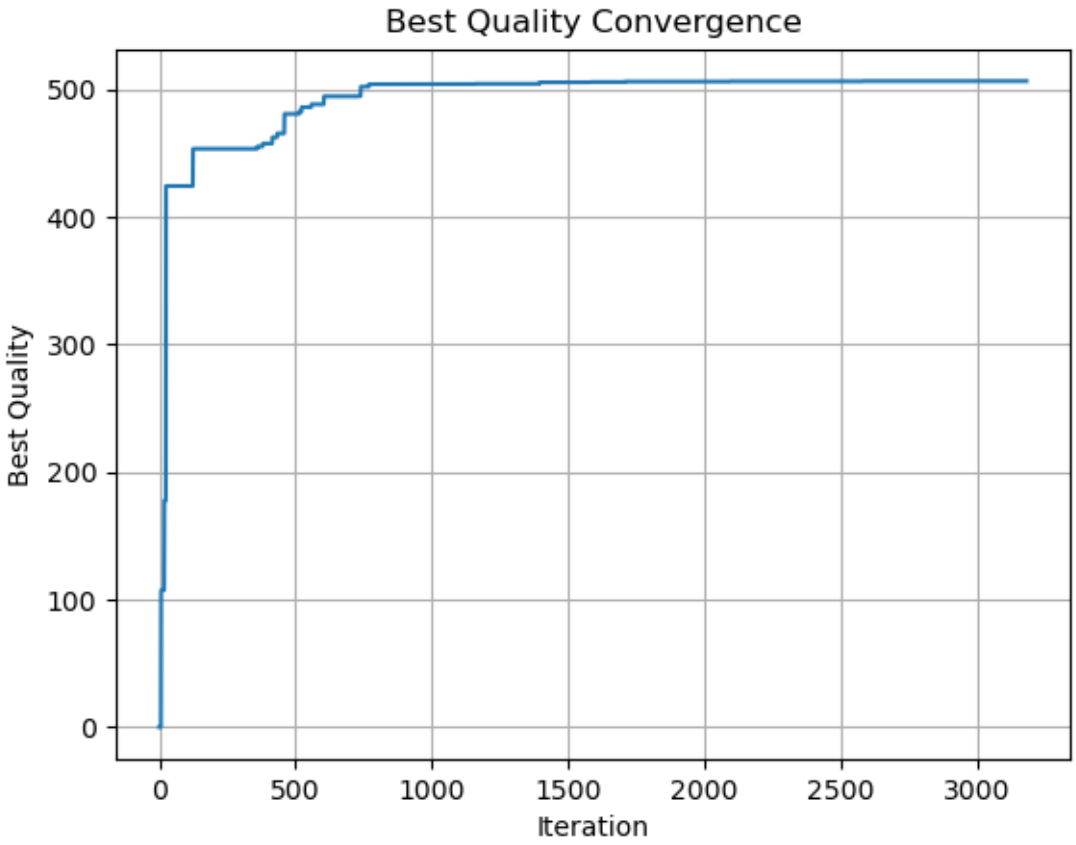

Figura 5: Convergência da *Best Quality*

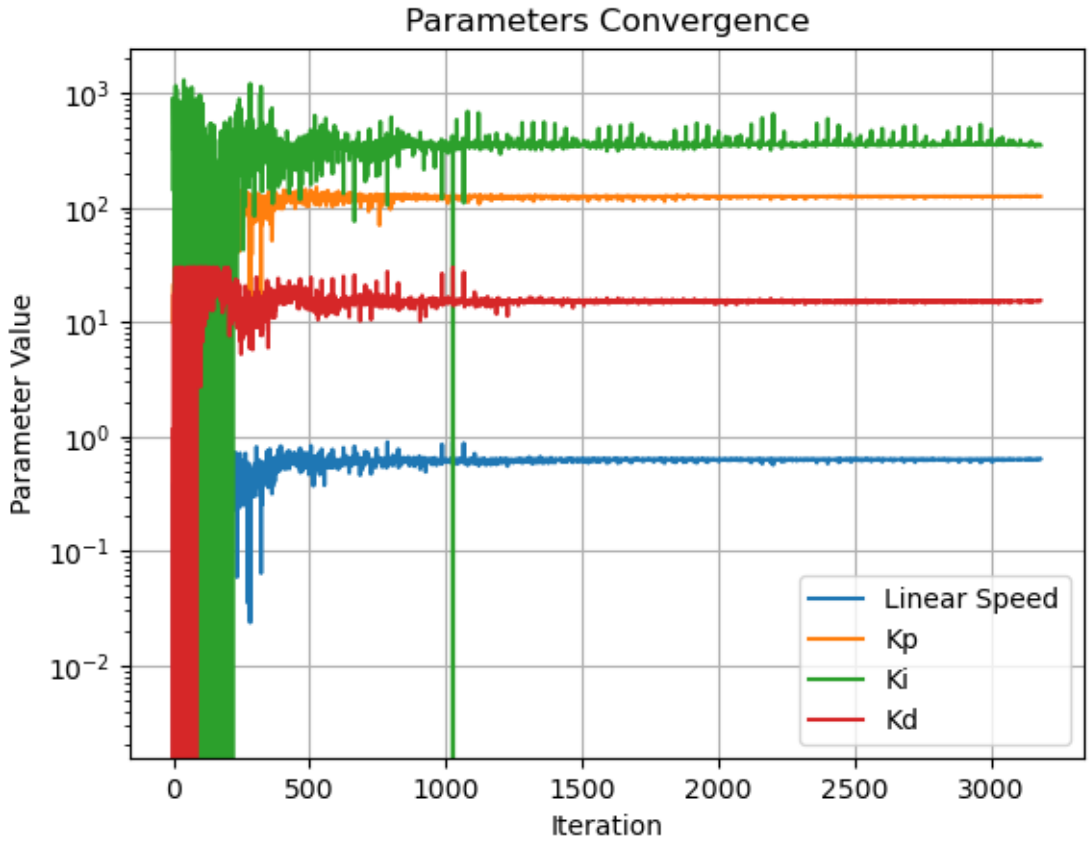

Figura 6: Convergência dos parâmetros $v,\ K_p,\ K_i\ e\ K_d$

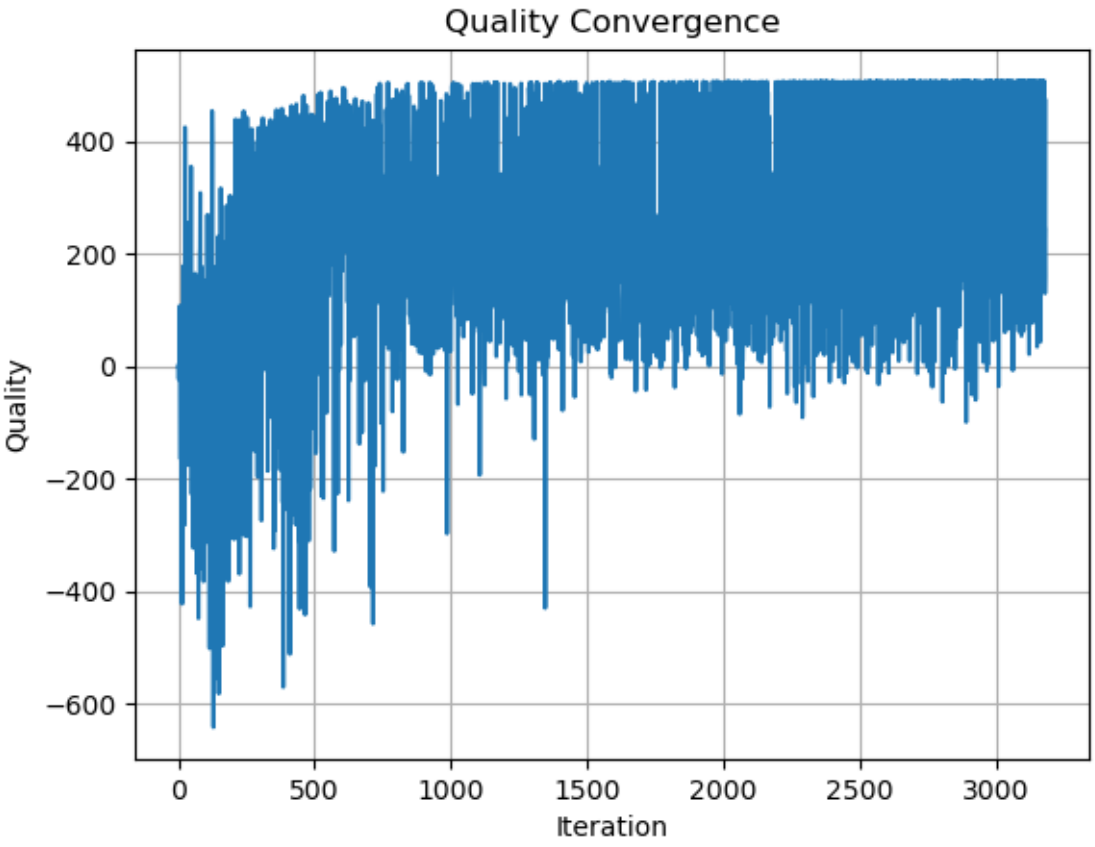

Figura 7: Convergência da *Quality* em função do número de iterações