```
# Partie 2: Régulation en distance

import rospy
from turtlesim.msg import Pose
from std_msgs.msg import Bool
from geometry_msgs.msg import Twist
from math import sqrt

pose = Pose()  # Variable globale pour stocker la pose de la tortue

def pose_callback(data):
    global pose
    pose = data

def calculate_distance(point1, point2):
    # Calcul de la distance euclidienne entre deux points
    return sqrt((point2.y - point1.y)**2 + (point2.x - point1.x)**2)

def main():
    rospy.init_node('regulation_distance_node')

    # Création des publishers et subscribers
    cmd_vel_pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)
    is_moving_pub = rospy.Publisher('/is_moving', Bool, queue_size=10)
    pose_sub = rospy.Subscriber('/turtle1/pose', Pose, pose_callback)

    rate = rospy.Rate(10)  # Fréquence de boucle (10 Hz)

    # Paramètres
    Kpl = 0.5  # Coefficient proportionnel pour la commande linéaire
    distance_tolerance = 0.1  # Seuil de tolérance de distance
    while not rospy.is_shutdown():
        # Calcul de l'erreur linéaire
        distance_error = calculate_distance(pose.position, waypoint)

        # Calcul de la commande linéaire
        linear_velocity = Kpl * distance_error

        if distance_error > distance_tolerance:
            # Publication de la commande linéaire et du statut de mouvement
            cmd_vel = Twist()
            cmd_vel.linear.x = linear_velocity
            cmd_vel_pub.publish(cmd_vel)
            is_moving_pub.publish(Bool(True))
        else:
            # La distance est inférieure au seuil de tolérance
            # Arrêt du mouvement et publication du statut de mouvement
            cmd_vel = Twist()
            cmd_vel_pub.publish(cmd_vel)
            is_moving_pub.publish(Bool(False))

        rate.sleep()

if __name__ == '__main__':
    try:
        main()
    except rospy.ROSInterruptException:
        pass
```