```python
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from rclpy.qos import QoSProfile, ReliabilityPolicy, HistoryPolicy
from sensor_msgs.msg import Image
from std_msgs.msg import String
from cv_bridge import CvBridge
import cv2
import numpy as np


class ColorObstacleDetector(Node):
    """
    Suscribe: /usb_cam/image_raw
    Publica : /obstaculos -> "rojo" | "verde" | "libre"
    - Sensibilidad reducida (HSV estricto, min_area alto, N frames consecutivos)
    - RetenciГіn: mantiene el Гєltimo color por 'hold_seconds' si desaparece
    """

    def __init__(self):
        super().__init__('color_obstacle_detector')

        # --- ParГЎmetros de sensibilidad ---
        self.declare_parameter('min_area', 1400)            # pГ-xeles mГ-nimos
        self.declare_parameter('consecutive_required', 3)   # N frames seguidos
        self.declare_parameter('hold_seconds', 1.0)         # retenciГіn tras perder color
        self.min_area = int(self.get_parameter('min_area').value)
        self.consecutive_required = int(self.get_parameter('consecutive_required').value)
        self.hold_seconds = float(self.get_parameter('hold_seconds').value)

        # --- Umbrales HSV mГЎs estrictos ---
        # Verde
        self.lower_green = np.array([45, 0, 0], dtype=np.uint8)
        self.upper_green = np.array([85, 255, 255], dtype=np.uint8)
        # Rojo (dos rangos)
        self.lower_red1 = np.array([0, 50, 50], dtype=np.uint8)
        self.upper_red1 = np.array([8, 255, 255], dtype=np.uint8)
        self.lower_red2 = np.array([170, 40, 40], dtype=np.uint8)
        self.upper_red2 = np.array([179, 255, 255], dtype=np.uint8)

        self.bridge = CvBridge()

        image_qos = QoSProfile(
            reliability=ReliabilityPolicy.BEST_EFFORT,
            history=HistoryPolicy.KEEP_LAST,
            depth=10,
        )

        self.image_sub = self.create_subscription(
            Image, '/usb_cam/image_raw', self.image_callback, image_qos
        )
        self.pub = self.create_publisher(String, '/obstaculos', 10)

        # Estado para "N consecutivos" y retenciГіn
        self.green_count = 0
        self.red_count = 0
        self.last_label = 'libre'
        self.last_detect_time = None  # segundos (float)

        self.get_logger().info(
            f'ColorObstacleDetector listo (min_area={self.min_area}, '
            f'consecutive_required={self.consecutive_required}, hold_seconds={self.hold_seconds}).'
        )

    def now_s(self) -> float:
        return self.get_clock().now().nanoseconds / 1e9

    def image_callback(self, msg: Image):
        try:
            frame = self.bridge.imgmsg_to_cv2(msg, desired_encoding='bgr8')
        except Exception as e:
            self.get_logger().error(f'CvBridge error: {e}')
            return

        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

        # MГЎscaras
        mask_green = cv2.inRange(hsv, self.lower_green, self.upper_green)
        mask_red = cv2.bitwise_or(
            cv2.inRange(hsv, self.lower_red1, self.upper_red1),
            cv2.inRange(hsv, self.lower_red2, self.upper_red2)
        )

        # MorfologГ-a fuerte para reducir ruido
        kernel = np.ones((7, 7), np.uint8)
        mask_green = cv2.morphologyEx(mask_green, cv2.MORPH_OPEN, kernel, iterations=2)
        mask_green = cv2.morphologyEx(mask_green, cv2.MORPH_CLOSE, kernel, iterations=2)
        mask_red = cv2.morphologyEx(mask_red, cv2.MORPH_OPEN, kernel, iterations=2)
        mask_red = cv2.morphologyEx(mask_red, cv2.MORPH_CLOSE, kernel, iterations=2)

        # ГЃreas mГЎximas
        green_area = self._max_contour_area(mask_green)
        red_area = self._max_contour_area(mask_red)

        green_ok = green_area >= self.min_area
        red_ok = red_area >= self.min_area

        # Debounce por frames consecutivos
        self.green_count = self.green_count + 1 if green_ok else 0
        self.red_count   = self.red_count + 1   if red_ok   else 0

        g_ready = self.green_count >= self.consecutive_required
        r_ready = self.red_count   >= self.consecutive_required

        decided_label = 'libre'
        now = self.now_s()

        if g_ready or r_ready:
            # DetecciГіn vГЎlida: elegir color y actualizar "Гєltimo visto"
            if g_ready and r_ready:
                decided_label = 'rojo' if red_area >= green_area else 'verde'
            elif r_ready:
                decided_label = 'rojo'
            else:
                decided_label = 'verde'
            self.last_label = decided_label
            self.last_detect_time = now
        else:
            # Sin detecciГіn actual: ВїaГєn dentro de la ventana de retenciГіn?
            if self.last_detect_time is not None and (now - self.last_detect_time) <= self.hold_seconds:
                decided_label = self.last_label  # mantener Гєltimo color
            else:
                decided_label = 'libre'
                self.last_label = 'libre'
                self.last_detect_time = None

        self.pub.publish(String(data=decided_label))
        self.get_logger().info(f"/obstaculos: {decided_label}  (red={red_area}, green={green_area})")

    @staticmethod
    def _max_contour_area(mask: np.ndarray) -> int:
        contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        return 0 if not contours else int(max(cv2.contourArea(c) for c in contours))


def main(args=None):
    rclpy.init(args=args)
    node = ColorObstacleDetector()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    finally:
        node.destroy_node()
        rclpy.shutdown()


if __name__ == '__main__':
    main()
```