Карта пооперационного контроля для участников и жюри по профилю «Робототехника»

№ п/п	Критерии оценивания	Макс. балл	Кол-во баллов, выставленных членами жюри		
1.	Общее оформление	0,25			
2.	Качество теоретического исследования	0,75			
3.	Разработка технологического процесса	3			
4.	Качество схем и чертежей	1			
5.	Обоснование выбора платформы	1			
6.	Креативность и новизна	1			
7.	Робототехническая сложность	7,5			
8.	Работоспособность	2,5			
9.	Эстетический вид	1			
10.	Трудоемкость	1			
11.	Практическая значимость	1			
12.	Презентация продукта	5			
	Итого	25			

По практическому туру максимальная оценка результатов участника возрастной группы (9-11 классы) определяется арифметической суммой всех баллов, полученных за выполнение заданий и не должна превышать 25 баллов.

Задание:

Максимальная оценка – 25 баллов.

Продолжительность этапа – 180 минут

```
* РОБОТ-СОРТИРОВЩИК ЦВЕТНЫХ КУБИКОВ
 *
 * Алгоритм:
 * 1. Старт с базы
 * 2. Последовательный сбор 3 кубиков (красный, синий, зеленый)
 * 3. Транспортировка каждого кубика в соответствующую цветовую зону
 * 4. Возврат на базу
 *
 * Используемые компоненты:
 * - Контроллер: Arduino Mega
 * - Датчик цвета: TCS3200 + Pixy2 для резервирования
 * - Двигатели: 4x DC motor with encoders
 * - Драйвер моторов: TB6612FNG
 * - Захват: сервопривод MG996R
 * - Дополнительно: Ультразвуковой датчик HC-SR04, IMU MPU-6050
 */

// ========== БИБЛИОТЕКИ И НАСТРОЙКИ ==========
#include <Servo.h>
#include <Wire.h>
#include <Pixy2.h>

// ========== КОНСТАНТЫ И ПЕРЕМЕННЫЕ ==========
// Пины моторов
#define MOTOR_L1 2
#define MOTOR_L2 3
#define MOTOR_R1 4
#define MOTOR_R2 5
```

```
#define PWM_L 6
#define PWM_R 7

// Пины энкодеров
#define ENCODER_L_A 18
#define ENCODER_L_B 19
#define ENCODER_R_A 20
#define ENCODER_R_B 21

// Пины сервопривода захвата
#define SERVO_PIN 9

// Пины датчиков цвета
#define S0 22
#define S1 23
#define S2 24
#define S3 25
#define COLOR_SENSOR_OUT 26

// Пины УЗ датчика
#define TRIG_PIN 28
#define ECHO_PIN 29

// Целевые цвета (калибровочные значения)
struct RGB {
  int red;
  int green;
  int blue;
};

RGB RED_CUBE = {120, 60, 60};
RGB BLUE_CUBE = {60, 70, 120};
RGB GREEN_CUBE = {70, 110, 70};
RGB RED_ZONE = {130, 70, 70};
RGB BLUE_ZONE = {70, 80, 130};
```

```
RGB GREEN_ZONE = {80, 120, 80};

// Координаты объектов (условные единицы)
struct Point {
  float x;
  float y;
};

Point BASE = {0, 0};
Point RED_CUBE_POS = {100, 50};
Point BLUE_CUBE_POS = {150, 0};
Point GREEN_CUBE_POS = {100, -50};
Point RED_ZONE_POS = {200, 80};
Point BLUE_ZONE_POS = {200, 0};
Point GREEN_ZONE_POS = {200, -80};

// ========== ОБЪЕКТЫ И ПЕРЕМЕННЫЕ ==========
Servo grabServo;
Pixy2 pixy;

// Текущая позиция и ориентация
Point currentPosition = {0, 0};
float currentAngle = 0;

// Состояния конечного автомата
enum RobotState {
  START,
  MOVE_TO_RED_CUBE,
  GRAB_RED_CUBE,
  MOVE_TO_RED_ZONE,
  RELEASE_RED_CUBE,
  MOVE_TO_BLUE_CUBE,
  GRAB_BLUE_CUBE,
  MOVE_TO_BLUE_ZONE,
  RELEASE_BLUE_CUBE,
```

```
  MOVE_TO_GREEN_CUBE,
  GRAB_GREEN_CUBE,
  MOVE_TO_GREEN_ZONE,
  RELEASE_GREEN_CUBE,
  RETURN_TO_BASE,
  MISSION_COMPLETE
};

RobotState currentState = START;

// ПИД параметры
float Kp = 0.8, Ki = 0.01, Kd = 0.1;
float prevError = 0;
float integral = 0;

// ========== ПРОТОТИПЫ ФУНКЦИЙ ==========
void setup();
void loop();
void stateMachine();
RGB readColorSensor();
float calculateColorDistance(RGB color1, RGB color2);
void moveToPoint(Point target);
void rotateToAngle(float targetAngle);
void grabCube();
void releaseCube();
bool detectObstacle();
void updatePosition();
void motorControl(int leftSpeed, int rightSpeed);
int calculatePID(float setpoint, float currentValue);
void updateEncoders();

// ========== НАСТРОЙКА ==========
void setup() {
  Serial.begin(9600);
```

```
// Инициализация моторов
pinMode(MOTOR_L1, OUTPUT);
pinMode(MOTOR_L2, OUTPUT);
pinMode(MOTOR_R1, OUTPUT);
pinMode(MOTOR_R2, OUTPUT);
pinMode(PWM_L, OUTPUT);
pinMode(PWM_R, OUTPUT);

// Инициализация энкодеров
pinMode(ENCODER_L_A, INPUT);
pinMode(ENCODER_L_B, INPUT);
pinMode(ENCODER_R_A, INPUT);
pinMode(ENCODER_R_B, INPUT);
attachInterrupt(digitalPinToInterrupt(ENCODER_L_A), updateEncoders, CHANGE);
attachInterrupt(digitalPinToInterrupt(ENCODER_R_A), updateEncoders, CHANGE);

// Инициализация сервопривода
grabServo.attach(SERVO_PIN);
grabServo.write(0); // начальное положение - разжато

// Инициализация датчиков
pinMode(S0, OUTPUT);
pinMode(S1, OUTPUT);
pinMode(S2, OUTPUT);
pinMode(S3, OUTPUT);
pinMode(COLOR_SENSOR_OUT, INPUT);
digitalWrite(S0, HIGH);
digitalWrite(S1, LOW);

pinMode(TRIG_PIN, OUTPUT);
pinMode(ECHO_PIN, INPUT);

// Инициализация камеры Pixy2
pixy.init();
```

```
  Serial.println("Робот инициализирован. Начинаем миссию!");
}

// ========== ГЛАВНЫЙ ЦИКЛ ==========
void loop() {
  stateMachine();
  updatePosition();
  delay(50); // основной цикл управления
}

// ========== КОНЕЧНЫЙ АВТОМАТ ==========
void stateMachine() {
  static unsigned long lastStateTime = 0;
  unsigned long currentTime = millis();

  switch(currentState) {
    case START:
      Serial.println("Состояние: START -> движение к красному кубику");
      currentState = MOVE_TO_RED_CUBE;
      break;

    case MOVE_TO_RED_CUBE:
      moveToPoint(RED_CUBE_POS);
      if (calculateDistance(currentPosition, RED_CUBE_POS) < 5) {
        RGB detectedColor = readColorSensor();
        if (calculateColorDistance(detectedColor, RED_CUBE) < 50) {
          Serial.println("Красный кубик обнаружен!");
          currentState = GRAB_RED_CUBE;
          lastStateTime = currentTime;
        }
      }
      break;

    case GRAB_RED_CUBE:
      if (currentTime - lastStateTime > 1000) {
```

```
    grabCube();
    Serial.println("Красный кубик захвачен!");
    currentState = MOVE_TO_RED_ZONE;
   }
   break;

 case MOVE_TO_RED_ZONE:
   moveToPoint(RED_ZONE_POS);
   if (calculateDistance(currentPosition, RED_ZONE_POS) < 5) {
    RGB zoneColor = readColorSensor();
    if (calculateColorDistance(zoneColor, RED_ZONE) < 50) {
     Serial.println("Красная зона достигнута!");
     currentState = RELEASE_RED_CUBE;
     lastStateTime = currentTime;
    }
   }
   break;

 case RELEASE_RED_CUBE:
   if (currentTime - lastStateTime > 1000) {
    releaseCube();
    Serial.println("Красный кубик размещен!");
    currentState = MOVE_TO_BLUE_CUBE;
   }
   break;

 // Аналогичные состояния для синего и зеленого кубиков...
 case MOVE_TO_BLUE_CUBE:
   moveToPoint(BLUE_CUBE_POS);
   // ... аналогично красному
   break;

 case MOVE_TO_BLUE_ZONE:
   // ... аналогично красному
   break;
```

```
    case MOVE_TO_GREEN_CUBE:
      moveToPoint(GREEN_CUBE_POS);
      // ... аналогично красному
      break;

    case MOVE_TO_GREEN_ZONE:
      // ... аналогично красному
      break;

    case RETURN_TO_BASE:
      moveToPoint(BASE);
      if (calculateDistance(currentPosition, BASE) < 5) {
        Serial.println("Миссия завершена! Робот вернулся на базу.");
        currentState = MISSION_COMPLETE;
        motorControl(0, 0); // полная остановка
      }
      break;

    case MISSION_COMPLETE:
      // Мигание светодиодом или сигнал завершения
      if (currentTime % 1000 < 500) {
        digitalWrite(LED_BUILTIN, HIGH);
      } else {
        digitalWrite(LED_BUILTIN, LOW);
      }
      break;
  }
}

// ========== ФУНКЦИИ ДВИЖЕНИЯ ==========
void moveToPoint(Point target) {
  // Вычисление угла к цели
  float targetAngle = atan2(target.y - currentPosition.y,
                            target.x - currentPosition.x) * 180 / PI;
```

```
  // Поворот к цели
  rotateToAngle(targetAngle);

  // Движение вперед с ПИД-регулированием
  float distance = calculateDistance(currentPosition, target);
  float error = distance;

  int baseSpeed = 150;
  int correction = calculatePID(0, error);

  motorControl(baseSpeed - correction, baseSpeed + correction);

  // Обнаружение препятствий
  if (detectObstacle()) {
    motorControl(0, 0);
    // Обход препятствия...
  }
}

void rotateToAngle(float targetAngle) {
  float angleError = targetAngle - currentAngle;

  // Нормализация ошибки угла
  while (angleError > 180) angleError -= 360;
  while (angleError < -180) angleError += 360;

  int rotationSpeed = calculatePID(0, angleError);
  motorControl(-rotationSpeed, rotationSpeed);

  // Ждем завершения поворота
  while (abs(angleError) > 2) {
    delay(10);
    angleError = targetAngle - currentAngle;
  }
```

```
}

// ========== ФУНКЦИИ ЗАХВАТА ==========
void grabCube() {
  for (int pos = 0; pos <= 90; pos += 2) {
    grabServo.write(pos);
    delay(15);
  }
  delay(500);
}

void releaseCube() {
  for (int pos = 90; pos >= 0; pos -= 2) {
    grabServo.write(pos);
    delay(15);
  }
  delay(500);
}

// ========== РАБОТА С ДАТЧИКАМИ ==========
RGB readColorSensor() {
  RGB color;

  // Чтение красной компоненты
  digitalWrite(S2, LOW);
  digitalWrite(S3, LOW);
  color.red = pulseIn(COLOR_SENSOR_OUT, LOW);

  // Чтение зеленой компоненты
  digitalWrite(S2, HIGH);
  digitalWrite(S3, HIGH);
  color.green = pulseIn(COLOR_SENSOR_OUT, LOW);

  // Чтение синей компоненты
  digitalWrite(S2, LOW);
```

```
  digitalWrite(S3, HIGH);
  color.blue = pulseIn(COLOR_SENSOR_OUT, LOW);

  return color;
}

float calculateColorDistance(RGB color1, RGB color2) {
  // Евклидово расстояние в RGB пространстве
  return sqrt(pow(color1.red - color2.red, 2) +
              pow(color1.green - color2.green, 2) +
              pow(color1.blue - color2.blue, 2));
}

bool detectObstacle() {
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  long duration = pulseIn(ECHO_PIN, HIGH);
  float distance = duration * 0.034 / 2;

  return (distance < 15); // препятствие ближе 15 см
}

// ========== СИСТЕМА КООРДИНАТ И ОДОМЕТРИЯ ==========
void updatePosition() {
  static long lastLeftTicks = 0, lastRightTicks = 0;
  static unsigned long lastUpdateTime = 0;

  // Здесь должна быть реализация одометрии на основе энкодеров
  // и IMU для коррекции дрейфа

  // Псевдокод для обновления позиции:
```

```
  // 1. Получить текущие тики энкодеров
  // 2. Вычислить пройденное расстояние
  // 3. Обновить currentPosition и currentAngle
}

float calculateDistance(Point p1, Point p2) {
  return sqrt(pow(p2.x - p1.x, 2) + pow(p2.y - p1.y, 2));
}

// ========== ПИД-РЕГУЛЯТОР ==========
int calculatePID(float setpoint, float currentValue) {
  float error = setpoint - currentValue;
  integral += error;
  float derivative = error - prevError;
  prevError = error;

  int output = Kp * error + Ki * integral + Kd * derivative;
  return constrain(output, -255, 255);
}

// ========== УПРАВЛЕНИЕ МОТОРАМИ ==========
void motorControl(int leftSpeed, int rightSpeed) {
  leftSpeed = constrain(leftSpeed, -255, 255);
  rightSpeed = constrain(rightSpeed, -255, 255);

  // Левый мотор
  if (leftSpeed > 0) {
    digitalWrite(MOTOR_L1, HIGH);
    digitalWrite(MOTOR_L2, LOW);
  } else {
    digitalWrite(MOTOR_L1, LOW);
    digitalWrite(MOTOR_L2, HIGH);
  }

  // Правый мотор
```

```
  if (rightSpeed > 0) {
    digitalWrite(MOTOR_R1, HIGH);
    digitalWrite(MOTOR_R2, LOW);
  } else {
    digitalWrite(MOTOR_R1, LOW);
    digitalWrite(MOTOR_R2, HIGH);
  }

  analogWrite(PWM_L, abs(leftSpeed));
  analogWrite(PWM_R, abs(rightSpeed));
}

// ========== ОБРАБОТЧИКИ ПРЕРЫВАНИЙ ДЛЯ ЭНКОДЕРОВ ==========
void updateEncoders() {
  // Реализация подсчета тиков энкодеров
  // Обновление глобальных переменных с тиками
}
```