§4 Заключительный этап: командная часть

Постановка задачи: Осуществить транспортную перевозку по суше, морю и воздуху, ориентируясь на показания датчиков и сигналы элементов инфраструктуры транспортной системы, а также настроить спутник для передачи радиосигнала и реализовать его позиционирование в пространстве. Команда получает оценку за совокупность решений и вольна выбирать любое сочетание конструкторов и соответствующих задач. Каждый из этих конструкторов должен быть собран участниками из предоставленных компонентов, протестирован и запрограммирован в соответствии с поставленной задачей.

Команде предоставляется **базовый набор для конструирования**:

- Набор для конструирования беспилотного автомобиля — 1.
- Набор для конструирования беспилотного катамарана — 1.
- Набор для конструирования коптера — 1.
- Набор для конструирования спутника связи — 1.

И следующий **инструментарий**:

- Ноутбук, предоставляемый организаторами – 1 (можно использовать свои).
- Паяльная станция – 1.
- Мультиметр — 1.
- Источники питания.
- Набор ручного инструмента (бокорезы, пинцет, кусачки, длинногубцы, макетный нож).
- Наборов отверток.
- Комплект расходных материалов (флюс, припой, суперклей, пинцет, изолента, пенопласт).
- ПО: Среда программирования Arduino IDE.

Командная часть заключительного этапа имеет **продолжительность** 3 дня (всего 18 астрономических часов), которые включают работу на стенде, подготовку, пробные заезды на полигоне, зачетные попытки на полигоне.

Команда может выбирать любой из подтреков, а также выполнять задания всех подтреков одновременно.

Подведение итогов: итоговый результат определяется суммированием лучших попыток в каждом из четырех подтреков. В каждом подтреке можно заработать 25 баллов

(уточнение далее). Максимальное количество баллов - 100. Количество зачетных попыток ограничено (каждый день - не более пяти по каждому подтреку), но при использовании каждой последующей попытки результат данной попытки уменьшается на 0.5 балла.

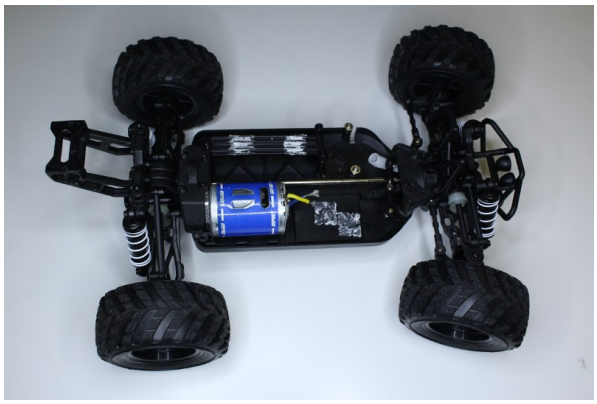
Тренировки: В первой половине (половина общего времени) дня свободный подход к стенду, в случае очереди – не более 5 минут на тестирование одной команде.

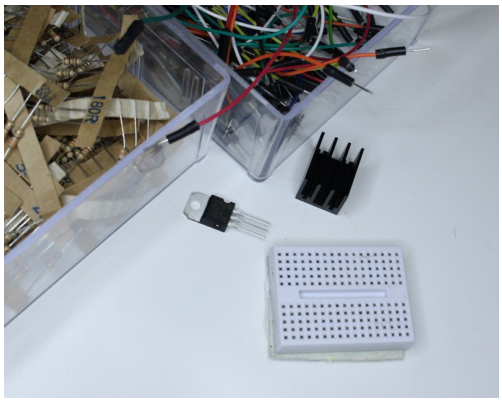
Оценка работ производится независимым образом: судья берет у команды готовый к испытанию аппарат (беспилотный автомобиль, беспилотный катамаран, коптер или спутник связи) и передавал его жюри, которое производило запуск аппарата на стенде и оценку работы согласно заданным критериям (см. далее).

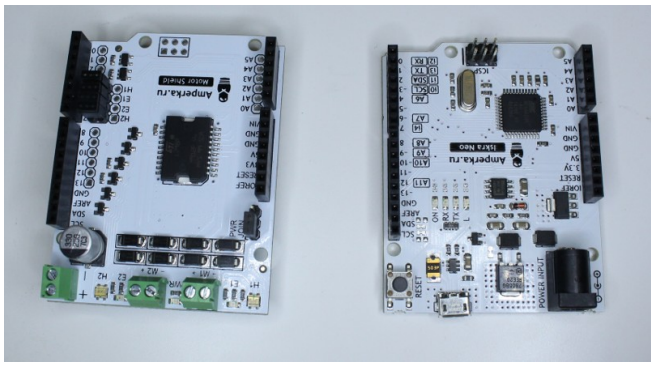
4.1. Подтрек «AutoNet»

В задаче по беспилотным автомобилям нужно будет разместить электронику и сенсоры на четырехколесном шасси (с автомобильной кинематикой) для решения задачи автономного передвижения по городу из одной точки в другую, ориентируясь по дорожной разметке и сигналам ИК-светофора.

На рисунках показан конструктор беспилотного автомобиля в виде компонентов:





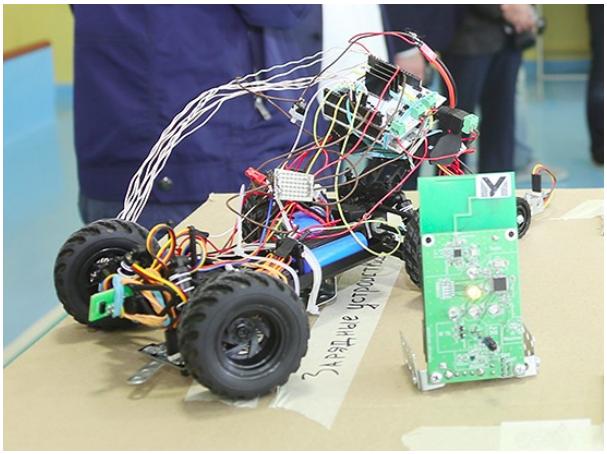


Базовый **набор для конструирования** (схемы, инструкции и т.д.):

- Шасси с мотором и сервоприводом - 1 шт.

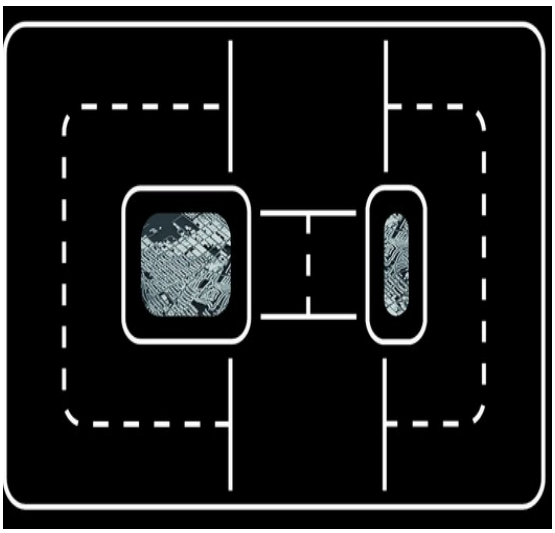
- Зарядное устройство - 1 шт.
- Управляющая плата.
- Плата Iskra Neo - 1 шт.
- Motor Shield - 1 шт.
- Провод USB - 1 шт.
- Набор датчиков линии (10 шт).
- Набор металлических деталей.

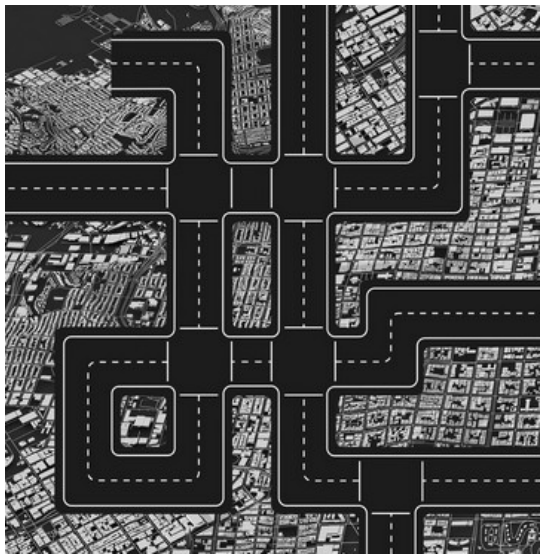
В собранном виде беспилотная машинка выглядит следующим образом (на фотографии изображена вместе с инфракрасным светофором):



Стенд (схема полигона):

- Баннерное покрытие с нанесенной разметкой (размер: 7м х 7м).
- ИК-светофоры.
- Терминал для управления светофорами.
- Радиомодуль (1 на все команды).



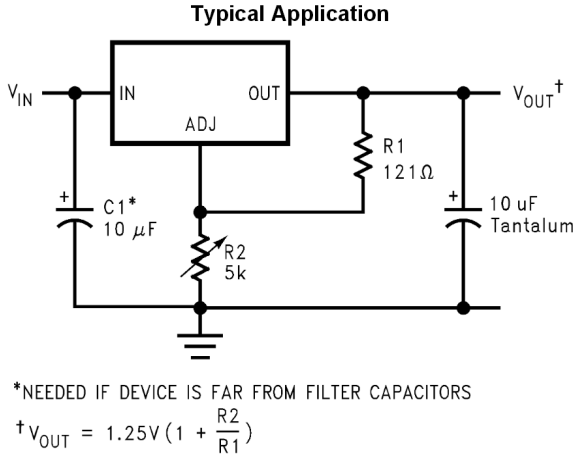


На рисунках показаны тренировочная и зачетная карты.

Перечень задач, решаемых в данном подтреке:

Задача 4.1.1. Сборка роботизированного автомобиля на базе четырёхколёсного шасси с автомобильной кинематикой. Для решения этой задачи участникам потребуются базовые навыки слесарных работ, работа с металлическим и другими видами крепежа.

Задача 4.1.2. Пайка системы стабилизации напряжения по схеме:



Для этого участникам потребуются: чтение электронной схемы, реализация её из электронных компонентов.

Задача 4.1.3. Создание единой схемы из платы Arduino, MotorShield, мотора и аккумулятора. Программирование микроконтроллера для управления мотором и сервоприводом на языке C/C++.

Задача 4.1.4. Создание системы сенсоров для ориентации робоавтомобиля в пространстве. Написание программы для сбора данных с сенсоров. Реализация езды по линии и езды в соответствии с данными ИК-светофора. Приём и декодирование ИК-сигнала.

Для решение этой задачи от учащихся потребуются базовые навыки слесарных работ, работа с металлическим и другими видами крепежа, чтение электронной схемы.

Задача 4.1.5. Реализация алгоритмов остановки на стоп-линии, поворота направо и налево по сигналу ИК-светофора.

А также набор **содержательных задач**:

- обработка данных сенсорной системы;
- алгоритм управления беспилотным транспортом с «автомобильной» кинематикой по данным сенсорной системы;
- алгоритм управления беспилотным транспортом с «автомобильной» кинематикой «по координатам» (расчет поворота сервомотора рулевой пары);

- расчет оптимальной скорости движения беспилотника в соответствии с характеристиками ходового двигателя (число оборотов, крутящий момент, параметры редуктора) для выполнения задач: движение по прямой, движение по дуге заданного радиуса, движение параллельно разметке
- определение оптимального расположения датчиков сенсорной системы, расчет интенсивности принимаемого сенсорами сигнала, определение пороговых значений.

Решение:

Пример решения задачи на языке C++:

```
// bodysensors.h
#pragma once

#include <stddef.h>
#include <stdint.h>
#include <string.h>
#include <math.h>

class BorderSensors
{
  public:
    static const size_t MAX_COUNT = 8;

  public:
    BorderSensors(const uint8_t *pins, size_t count) :
      _pins(),
      _data(),
      _count(count),
      _error(NAN),
      _max_error()
    {
      if (_count > MAX_COUNT) {
        _count = MAX_COUNT;
      }

      _max_error = static_cast<float>(_count - 1) / 2;

      memcpy(_pins, pins, _count);
      memset(_data, 0, _count * sizeof(int));
    }

    void init();

    void update();

    size_t getCount()
    {
      return _count;
    }
```

```
40.
41.     void getData(int *data)
42.     {
43.       memcpy(data, _data, _count * sizeof(int));
44.     }
45.
46.     float getMaxError()
47.     {
48.       return _max_error;
49.     }
50.
51.     float getError()
52.     {
53.       return _error;
54.     }
55.
56.   private:
57.     uint8_t _pins[MAX_COUNT];
58.     int _data[MAX_COUNT];
59.     size_t _count;
60.     float _error;
61.     float _max_error;
62. };
63.
64. // bodysensors.cpp
65. #include "bordersensors.h"
66.
67. #include <limits.h>
68.
69. #include <Arduino.h>
70.
71. const int WHITE_THRESHOLD = 75;
72. const int BLACK_THRESHOLD = 225;
73.
74. void BorderSensors::init()
75. {
76. }
77.
78. void BorderSensors::update()
79. {
80.   for (size_t i = 0; i < _count; i++) {
81.     _data[i] = analogRead(_pins[i]);
82.   }
83.
84.   int max_value = INT_MIN;
85.   int min_value = INT_MAX;
86.   size_t min_position;
87.
88.   for (size_t i = 0; i < _count; i++) {
89.     if (_data[i] >= max_value) {
90.       max_value = _data[i];
```

```
91.     }
92.
93.     if (_data[i] <= min_value) {
94.       min_value = _data[i];
95.       min_position = i;
96.     }
97.   }
98.
99.   if (min_value >= BLACK_THRESHOLD) {
100.    _error = NAN;
101.    return;
102.  }
103.
104.  if (_count < 2) {
105.    _error = 0.0f;
106.    return;
107.  }
108.
109.  if (max_value <= WHITE_THRESHOLD) {
110.    _error = NAN;
111.    return;
112.  }
113.
114.  int next_to_min_value;
115.  size_t next_to_min_position;
116.
117.  if (min_position == 0) {
118.    next_to_min_value = _data[1];
119.    next_to_min_position = 1;
120.  } else if (min_position == (_count - 1)) {
121.    next_to_min_value = _data[_count - 2];
122.    next_to_min_position = _count - 2;
123.  } else if (_data[min_position - 1] < _data[min_position + 1]) {
124.    next_to_min_value = _data[min_position - 1];
125.    next_to_min_position = min_position - 1;
126.  } else {
127.    next_to_min_value = _data[min_position + 1];
128.    next_to_min_position = min_position + 1;
129.  }
130.
131.  if (next_to_min_value > BLACK_THRESHOLD) {
132.    next_to_min_value = BLACK_THRESHOLD;
133.  }
134.  if (next_to_min_value < WHITE_THRESHOLD) {
135.    next_to_min_value = WHITE_THRESHOLD;
136.  }
137.  if (min_value < WHITE_THRESHOLD) {
138.    min_value = WHITE_THRESHOLD;
139.  }
140.  next_to_min_value -= WHITE_THRESHOLD;
141.  min_value -= WHITE_THRESHOLD;
```

```
142.
143.   float line_position = min_position;
144.   float correction = 0.5f * ((BLACK_THRESHOLD - WHITE_THRESHOLD) - next_to_min_value) /
145.                             ((BLACK_THRESHOLD - WHITE_THRESHOLD) - min_value);
146.   if (next_to_min_position > min_position) {
147.     line_position += correction;
148.   } else {
149.     line_position -= correction;
150.   }
151.
152.   float center_position = _max_error;
153.
154.   _error = line_position - center_position;
155. }
156.
157. // irreceiver.h
158. #pragma once
159.
160. #include <stdint.h>
161.
162. class IrReceiver
163. {
164.   public:
165.     IrReceiver(uint8_t pin) :
166.       _pin(pin),
167.       _state(STATE_BEGIN),
168.       _data(0),
169.       _bit_counter(0)
170.     {
171.     }
172.
173.     void init();
174.
175.     void handleEdge();
176.
177.     void handleTimeout();
178.
179.     bool isDataReceived();
180.
181.     uint16_t getReceivedData();
182.
183.   private:
184.     enum State
185.     {
186.       STATE_BEGIN,
187.       STATE_START,
188.       STATE_DATA_STAGE_1,
189.       STATE_DATA_STAGE_2,
190.       STATE_CHECKING,
191.       STATE_RECEIVED
192.     };
```

```
193.
194.   private:
195.     uint8_t _pin;
196.     volatile State _state;
197.     volatile uint16_t _data;
198.     volatile int _bit_counter;
199. };
200.
201. // irreceiver.cpp
202. #include "irreceiver.h"
203.
204. #include <Arduino.h>
205.
206. const unsigned int START_TIMEOUT = 2500;
207. const unsigned int DATA_STAGE_1_TIMEOUT = 1200;
208. const unsigned int DATA_STAGE_2_TIMEOUT = 900;
209. const unsigned int CHECKING_TIMEOUT = 10000;
210.
211. static IrReceiver *active_receiver = NULL;
212.
213. static void initTimer()
214. {
215.   TCCR4A = 0;
216.   TCCR4B = 0;
217.   TCCR4C = 0;
218.   TCCR4D = 0;
219.   TCCR4E = 0;
220. }
221.
222. static void startTimer(unsigned int us)
223. {
224.   TC4H = 0;
225.   TCNT4 = 0;
226.   OCR4C = microsecondsToClockCycles(us) / 1024;
227.   TCCR4B = (1 << PSR4) |
228.            (1 << CS43) | (0 << CS42) |
229.            (1 << CS41) | (1 << CS40);
230.   TIMSK4 = (1 << TOIE4);
231. }
232.
233. static void stopTimer()
234. {
235.   TIMSK4 = 0;
236.   TCCR4B = 0;
237. }
238.
239. static void edgeHandler()
240. {
241.   if (active_receiver != NULL) {
242.     active_receiver->handleEdge();
243.   }
```

```
244. }
245.
246. ISR(TIMER4_OVF_vect)
247. {
248.   if (active_receiver != NULL) {
249.     active_receiver->handleTimeout();
250.   }
251. }
252.
253. void IrReceiver::init()
254. {
255.   _state = STATE_BEGIN;
256.
257.   initTimer();
258.
259.   active_receiver = this;
260.
261.   pinMode(_pin, INPUT);
262.
263.   attachInterrupt(digitalPinToInterrupt(_pin),
264.                   edgeHandler,
265.                   FALLING);
266. }
267.
268. void IrReceiver::handleEdge()
269. {
270.   stopTimer();
271.
272.   if (_state == STATE_BEGIN) {
273.     startTimer(START_TIMEOUT);
274.
275.     _state = STATE_START;
276.     return;
277.   }
278.
279.   if (_state == STATE_START) {
280.     _state = STATE_BEGIN;
281.     return;
282.   }
283.
284.   if (_state == STATE_DATA_STAGE_1) {
285.     startTimer(DATA_STAGE_2_TIMEOUT);
286.
287.     _state = STATE_DATA_STAGE_2;
288.     return;
289.   }
290.
291.   if (_state == STATE_DATA_STAGE_2) {
292.     _state = STATE_BEGIN;
293.     return;
294.   }
```

```
295.
296.   if (_state == STATE_CHECKING) {
297.     _state = STATE_BEGIN;
298.     return;
299.   }
300. }
301.
302. void IrReceiver::handleTimeout()
303. {
304.   stopTimer();
305.
306.   if (_state == STATE_START) {
307.     if (digitalRead(_pin) == HIGH) {
308.       startTimer(DATA_STAGE_1_TIMEOUT);
309.       _data = 0;
310.       _bit_counter = 12;
311.
312.       _state = STATE_DATA_STAGE_1;
313.     } else {
314.       _state = STATE_BEGIN;
315.     }
316.     return;
317.   }
318.
319.   if (_state == STATE_DATA_STAGE_1) {
320.     _state = STATE_BEGIN;
321.     return;
322.   }
323.
324.   if (_state == STATE_DATA_STAGE_2) {
325.     if (digitalRead(_pin) == HIGH) {
326.       _data <<= 1;
327.     } else {
328.       _data <<= 1;
329.       _data |= 1;
330.     }
331.
332.     _bit_counter--;
333.
334.     if (_bit_counter > 0) {
335.       startTimer(DATA_STAGE_1_TIMEOUT);
336.
337.       _state = STATE_DATA_STAGE_1;
338.     } else {
339.       startTimer(CHECKING_TIMEOUT);
340.
341.       _state = STATE_CHECKING;
342.     }
343.     return;
344.   }
345.
```

```
346.   if (_state == STATE_CHECKING) {
347.     _state = STATE_RECEIVED;
348.     return;
349.   }
350. }
351.
352. bool IrReceiver::isDataReceived()
353. {
354.   cli();
355.   State state = _state;
356.   sei();
357.   return (state == STATE_RECEIVED);
358. }
359.
360. uint16_t IrReceiver::getReceivedData()
361. {
362.   if (isDataReceived()) {
363.     cli();
364.     uint16_t data = _data;
365.     _state = STATE_BEGIN;
366.     sei();
367.     return data;
368.   } else {
369.     return 0;
370.   }
371. }
372.
373.
374. // motor.h
375. #pragma once
376.
377. #include <stdint.h>
378.
379. class Motor
380. {
381.   public:
382.     static const int MAX_SPEED = 255;
383.     static const int START_SPEED = 192;
384.     static const int FORWARD_SPEED = 130;
385.     //static const int FORWARD_SPEED = 140;
386.     //static const int FORWARD_SPEED = 150;
387.     //static const int FORWARD_SPEED = 160;
388.     //static const int SEARCH_SPEED = 130;
389.     //static const int SEARCH_SPEED = 140;
390.     static const int SEARCH_SPEED = 160;
391.     //static const int SEARCH_SPEED = 160;
392.     //static const int LEFT_SPEED = 130;
393.     static const int LEFT_SPEED = 160;
394.     //static const int LEFT_SPEED = 150;
395.     //static const int LEFT_SPEED = 160;
396.     //static const int RIGHT_SPEED = 130;
```

```
397.     static const int RIGHT_SPEED = 160;
398.     //static const int RIGHT_SPEED = 150;
399.     //static const int RIGHT_SPEED = 160;
400.
401.   public:
402.     Motor(uint8_t speed_pin, uint8_t direction_pin) :
403.       _speed_pin(speed_pin),
404.       _direction_pin(direction_pin)
405.     {
406.     }
407.
408.     void init();
409.
410.     void setSpeed(int speed);
411.
412.   private:
413.     uint8_t _speed_pin;
414.     uint8_t _direction_pin;
415. };
416.
417. // motor.cpp
418. #include "motor.h"
419.
420. #include <Arduino.h>
421.
422. void Motor::init()
423. {
424.   digitalWrite(_direction_pin, LOW);
425.   pinMode(_direction_pin, OUTPUT);
426.
427.   digitalWrite(_speed_pin, LOW);
428.   pinMode(_speed_pin, OUTPUT);
429. }
430.
431. void Motor::setSpeed(int speed)
432. {
433.   if (speed >= 0) {
434.     digitalWrite(_direction_pin, HIGH);
435.   } else {
436.     digitalWrite(_direction_pin, LOW);
437.   }
438.
439.   speed = abs(speed);
440.   if (speed > 255) {
441.     speed = 255;
442.   }
443.
444.   analogWrite(_speed_pin, static_cast<uint8_t>(speed));
445. }
446.
447. // steering.h
```

```
448. #pragma once
449.
450. #include <stdint.h>
451.
452. #include <Servo.h>
453.
454. class Steering
455. {
456.   public:
457.     static const int MAX_ANGLE = 40;
458.
459.   public:
460.     Steering(uint8_t pin, int straight_position) :
461.       _pin(pin),
462.       _straight_position(straight_position)
463.     {
464.     }
465.
466.     void init();
467.
468.     void setAngle(int angle);
469.
470.   private:
471.     Servo _servo;
472.     uint8_t _pin;
473.     int _straight_position;
474. };
475.
476. //steering.cpp
477. #include "steering.h"
478.
479. #include <Arduino.h>
480.
481. void Steering::init()
482. {
483.   digitalWrite(_pin, LOW);
484.   pinMode(_pin, OUTPUT);
485.
486.   _servo.attach(_pin);
487.   _servo.write(_straight_position);
488. }
489.
490. void Steering::setAngle(int angle)
491. {
492.   if (angle > MAX_ANGLE) {
493.     angle = MAX_ANGLE;
494.   } else if (angle < -MAX_ANGLE) {
495.     angle = -MAX_ANGLE;
496.   }
497.
498.   _servo.write(_straight_position + angle);
```

```
499. }
500.
501.
502. // smartcar.ino
503.
504. #include <Servo.h>
505.
506. #include <stddef.h>
507. #include <stdint.h>
508. #include <string.h>
509.
510. #include "motor.h"
511. #include "steering.h"
512. #include "bordersensors.h"
513. #include "irreceiver.h"
514. #include "pidcontroller.h"
515.
516. const uint8_t MOTOR_DIRECTION_PIN = 4;
517. const uint8_t MOTOR_SPEED_PIN = 5;
518.
519. Motor motor(MOTOR_SPEED_PIN, MOTOR_DIRECTION_PIN);
520.
521. //const int SERVO_CENTER_OFFSET = 90;
522. const int SERVO_CENTER_OFFSET = 82;
523.
524. const uint8_t SERVO_PIN = 11;
525.
526. Steering steering(SERVO_PIN, SERVO_CENTER_OFFSET);
527.
528. const int BORDER_SENSOR_COUNT = 8;
529.
530. const uint8_t BORDER_SENSOR_PINS[BORDER_SENSOR_COUNT] =
531.     {10, 8, 5, 4, 3, 2, 1, 0};
532.
533. BorderSensors border_sensors(BORDER_SENSOR_PINS, BORDER_SENSOR_COUNT);
534.
535. const uint8_t FRONT_STOP_SENSOR_PIN = 11;
536. const uint8_t BACK_STOP_SENSOR_PIN = 9;
537.
538. const int FRONT_STOP_SENSOR_THRESHOLD = 100;
539. const int BACK_STOP_SENSOR_THRESHOLD = 100;
540.
541. const uint8_t IR_RECEIVER_PIN = 7;
542.
543. IrReceiver receiver(IR_RECEIVER_PIN);
544.
545. const uint8_t SIGNAL_STOP = 0xf0;
546. const uint8_t SIGNAL_FORWARD = 0xc0;
547. const uint8_t SIGNAL_LEFT = 0xa0;
548. const uint8_t SIGNAL_RIGHT = 0x90;
549.
```

```
550. const unsigned long LINE_SEARCH_TIMEOUT = 10000000;
551. const unsigned long LINE_CAPTURE_DELAY = 250000;
552.
553. PidController drive_controller(1.0f, -0.1f, -0.005f, 1000000);
554.
555. enum Direction
556. {
557.   DIRECTION_NONE,
558.   DIRECTION_FORWARD,
559.   DIRECTION_LEFT,
560.   DIRECTION_RIGHT
561. };
562.
563. Direction requested_direction = DIRECTION_FORWARD;
564. Direction hinted_direction = DIRECTION_FORWARD;
565.
566. bool wait_for_line = false;
567. bool delayed_capture = false;
568.
569. unsigned long forced_steering_timestamp = 0;
570.
571. typedef void (*State)();
572.
573. State _state;
574.
575. void switchState(State state)
576. {
577.   _state = state;
578. }
579.
580. void state_begin();
581. void state_pre_drive();
582. void state_drive();
583. void state_pre_stop();
584. void state_stop();
585. void state_forced_steering();
586. void state_pre_end();
587. void state_end();
588.
589. void state_begin()
590. {
591.   steering.setAngle(0);
592.
593.   switchState(state_pre_drive);
594. }
595.
596. void state_pre_drive()
597. {
598.   drive_controller.reset();
599.
600.   motor.setSpeed(Motor::START_SPEED);
```

```
601.
602.   delay(500);
603.
604.   motor.setSpeed(Motor::FORWARD_SPEED);
605.
606.   switchState(state_drive);
607. }
608.
609. void state_drive()
610. {
611.   if (receiver.isDataReceived()) {
612.     uint8_t signal = receiver.getReceivedData() >> 4;
613.
614.     if (signal == SIGNAL_FORWARD) {
615.       wait_for_line = true;
616.       requested_direction = DIRECTION_FORWARD;
617.     }
618.
619.     if (signal == SIGNAL_LEFT) {
620.       wait_for_line = true;
621.       requested_direction = DIRECTION_LEFT;
622.     }
623.
624.     if (signal == SIGNAL_RIGHT) {
625.       wait_for_line = true;
626.       requested_direction = DIRECTION_RIGHT;
627.     }
628.
629.     if (signal == SIGNAL_STOP) {
630.       wait_for_line = true;
631.       requested_direction = DIRECTION_NONE;
632.     }
633.   }
634.
635.   if (analogRead(FRONT_STOP_SENSOR_PIN) < FRONT_STOP_SENSOR_THRESHOLD) {
636.     if (wait_for_line) {
637.       wait_for_line = false;
638.
639.       if (requested_direction == DIRECTION_NONE) {
640.         switchState(state_pre_stop);
641.         return;
642.       }
643.
644.       forced_steering_timestamp = micros();
645.
646.       switchState(state_forced_steering);
647.       return;
648.     }
649.   }
650.
651.   border_sensors.update();
```

```
652.
653.   static bool seen_good = false;
654.   static bool seen_bad = false;
655.   static unsigned long last_good_timestamp;
656.   static unsigned long last_bad_timestamp;
657.
658.   float error = border_sensors.getError();
659.
660.   if (isnan(error)) {
661.     seen_bad = true;
662.     last_bad_timestamp = micros();
663.
664.     if (!seen_good) {
665.       switchState(state_pre_end);
666.       return;
667.     }
668.
669.     if (micros() - last_good_timestamp >= LINE_SEARCH_TIMEOUT) {
670.       switchState(state_pre_end);
671.       return;
672.     }
673.
674.     motor.setSpeed(Motor::SEARCH_SPEED);
675.
676.     if (hinted_direction == DIRECTION_FORWARD) {
677.       steering.setAngle(0);
678.     } else if (hinted_direction == DIRECTION_LEFT) {
679.       steering.setAngle(-Steering::MAX_ANGLE);
680.     } else if (hinted_direction == DIRECTION_RIGHT) {
681.       steering.setAngle(Steering::MAX_ANGLE);
682.     }
683.
684.     return;
685.   }
686.
687.   if (delayed_capture && seen_bad) {
688.     if (micros() - last_bad_timestamp < LINE_CAPTURE_DELAY) {
689.       return;
690.     } else {
691.       delayed_capture = false;
692.     }
693.   }
694.
695.   seen_good = true;
696.   last_good_timestamp = micros();
697.
698.   if (error >= 0.0f) {
699.     hinted_direction = DIRECTION_RIGHT;
700.   } else {
701.     hinted_direction = DIRECTION_LEFT;
702.   }
```

```
703.
704.   float max_error = border_sensors.getMaxError();
705.
706.   float reaction = drive_controller.react(error / max_error, micros());
707.
708.   int angle = static_cast<int>(truncf(reaction * Steering::MAX_ANGLE));
709.
710.   motor.setSpeed(Motor::FORWARD_SPEED);
711.
712.   steering.setAngle(angle);
713. }
714.
715. void state_pre_stop()
716. {
717.   steering.setAngle(0);
718.
719.   motor.setSpeed(-Motor::START_SPEED);
720.
721.   delay(1000);
722.
723.   motor.setSpeed(0);
724.
725.   switchState(state_stop);
726. }
727.
728. void state_stop()
729. {
730.   if (receiver.isDataReceived()) {
731.     uint8_t signal = receiver.getReceivedData() >> 4;
732.
733.     if (signal == SIGNAL_FORWARD) {
734.       requested_direction = DIRECTION_FORWARD;
735.       wait_for_line = true;
736.       switchState(state_pre_drive);
737.       return;
738.     }
739.
740.     if (signal == SIGNAL_LEFT) {
741.       requested_direction = DIRECTION_LEFT;
742.       wait_for_line = true;
743.       switchState(state_pre_drive);
744.       return;
745.     }
746.
747.     if (signal == SIGNAL_RIGHT) {
748.       requested_direction = DIRECTION_RIGHT;
749.       wait_for_line = true;
750.       switchState(state_pre_drive);
751.       return;
752.     }
753.   }
```

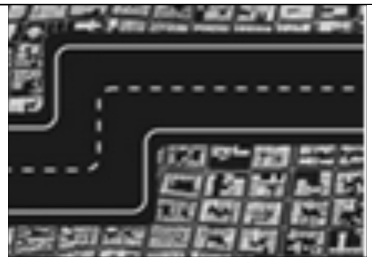
```
754. }
755.
756. void state_forced_steering()
757. {
758.   if (receiver.isDataReceived()) {
759.     receiver.getReceivedData();
760.   }
761.
762.   if (micros() - forced_steering_timestamp > LINE_CAPTURE_DELAY) {
763.     if (!wait_for_line) {
764.       if (requested_direction == DIRECTION_RIGHT) {
765.         wait_for_line = true;
766.       }
767.
768.       if ((requested_direction == DIRECTION_LEFT) ||
769.           (requested_direction == DIRECTION_FORWARD)) {
770.         if (analogRead(BACK_STOP_SENSOR_PIN) < BACK_STOP_SENSOR_THRESHOLD) {
771.           wait_for_line = true;
772.         }
773.       }
774.     }
775.   }
776.
777.   if (analogRead(FRONT_STOP_SENSOR_PIN) < FRONT_STOP_SENSOR_THRESHOLD) {
778.     if (wait_for_line) {
779.       wait_for_line = false;
780.
781.       drive_controller.reset();
782.
783.       if (requested_direction == DIRECTION_FORWARD) {
784.         //delayed_capture = true;
785.         //delay(500);
786.         delay(750);
787.       } else if (requested_direction == DIRECTION_RIGHT) {
788.         //delayed_capture = true;
789.         delay(500);
790.       } else if (requested_direction == DIRECTION_LEFT) {
791.         //delayed_capture = true;
792.         //delay(500);
793.       }
794.
795.       //hinted_direction = requested_direction;
796.
797.       hinted_direction = DIRECTION_RIGHT;
798.
799.       switchState(state_drive);
800.       return;
801.     }
802.   }
803.
804.   if (requested_direction == DIRECTION_FORWARD) {
```

```
805.     motor.setSpeed(Motor::FORWARD_SPEED);
806.
807.     steering.setAngle(0);
808.   } else if (requested_direction == DIRECTION_LEFT) {
809.     if (wait_for_line) {
810.       motor.setSpeed(Motor::LEFT_SPEED);
811.
812.       steering.setAngle(-Steering::MAX_ANGLE);
813.     } else {
814.       motor.setSpeed(Motor::FORWARD_SPEED);
815.
816.       steering.setAngle(0);
817.     }
818.   } else if (requested_direction == DIRECTION_RIGHT) {
819.     motor.setSpeed(Motor::RIGHT_SPEED);
820.
821.     steering.setAngle(Steering::MAX_ANGLE);
822.   }
823. }
824.
825. void state_pre_end()
826. {
827.   motor.setSpeed(0);
828.
829.   delay(500);
830.   steering.setAngle(-Steering::MAX_ANGLE);
831.   delay(500);
832.   steering.setAngle(Steering::MAX_ANGLE);
833.   delay(500);
834.   steering.setAngle(0);
835.
836.   switchState(state_end);
837. }
838.
839. void state_end()
840. {
841. }
842.
843. void setup()
844. {
845.   Serial.begin(115200);
846.   Serial1.begin(9600);
847.
848.   motor.init();
849.
850.   steering.init();
851.
852.   border_sensors.init();
853.
854.   receiver.init();
855.
```

```
856.   switchState(state_begin);
857. }
858.
859. void loop()
860. {
861.   _state();
862. }
```

4.1.6. Критерии оценки:

Элементы полигона и трассы:

	Название	**Описание**	
1	Проезжая часть с двусторонним движением (дорога)	Ровная горизонтальная поверхность черного цвета (ширина полотна 60 см), ограниченная сплошной разметкой. По середине дорожного полотна проходит разделительная полоса (прерывистая разметка)	
2	Прерывистая разметка	Белая линия, ширина 2 см. длина прерывания – 10 см. Состоит из прямолинейных участков.	
3	Сплошная разметка	Непрерывная белая линия (ширина 2 см). Состоит из прямолинейных и криволинейных (радиус кривизны задан и одинаков на всех участках трассы) участков.	
4	СТОП-линия	Участок прямой белой линии (ширина 2 см), расположенный перпендикулярно к линиям основной разметки, находится непосредственно перед перекрестком.	

5	Перекресток	Пересечение двух дорог дорожного полотна. Угол пересечения – 90 градусов. Внутри перекрестка отсутствует какая-либо разметка.	
6	ИК-светофор	Устройство, которое расположено на перекрестке, и в формате ИК-сигнала и световой индикации передает информацию о возможности движения в трех направлениях («направо», «прямо», «налево») либо «остановки»	 
7	«Жилая зона»	Остальное пространство полигона, по которому нельзя осуществлять перемещения транспортным средствам. Ограничено сплошной разметкой и обозначено схематической жилой застройкой.	

Порядок оценки и критерии:

	Действия устройства «робовтомобиль», оцениваемые жюри	**Описание действия, особенности расположения на полигоне и т.д.**	**Критерий выполнения действия**	**Баллы**
1.	Движение робоавтомобиля по прямой в течении 2 секунд	Оценка движения происходит на длинном прямолинейном участке проезжей части. Устройство устанавливается по центру дорожного полотна (на прерывистую разметку) параллельно линиям сплошной разметки.	Робоавтомобиль совершает прямолинейное движение в течении двух секунд, Во время движения устройство целиком остается на «проезжей части»	3
2.	Поворот «НАПРАВО	Оценка выполнения действия происходит на любой части	Робоавтомобиль совершает движение по прямой в течении	1

		полигона. После движения прямо в течении 1 секунды, устройство должно совершать поворот направо в течении 1 секунды.	1 секунды, затем поворот направо в течении 1 секунды	
	Поворот «НАЛЕВО	Оценка выполнения действия происходит на любой части полигона. После движения прямо в течении 1 секунды, устройство должно совершать поворот налево в течении 1 секунды.	Робоавтомобиль совершает движение по прямой в течении 1 секунды, затем поворот налево в течении 1 секунды	1
	«Слалом»	Оценка выполнения действия происходит на любой части полигона. После движения прямо в течении 1 секунды, устройство должно совершать поворот направо в течении 1 секунды затем совершать поворот налево в течении 1 секунды.	Робоавтомобиль совершает движение по прямой в течении 1 секунды, затем поворот направо в течении 1 секунды, затем налево в течении 1 секунды	1
3.	Движение по разметке	Оценка движения происходит на любом отрезке проезжей части, содержащей криволинейные участки. Устройство устанавливается по центру одной полосы дорожного полотна (полоса ограничена сплошной линией справа и прерывистой слева). Начало и конец отрезка определяются жюри для каждой попытки.	Робоавтомобиль совершает движение по заданному отрезку проезжей части, не одним своим колесом не выходя за пределы полосы, по которой движется	4
4.	Обработка ИК-сигнала светофора	Оценка данного действия происходит на рабочем месте с использованием компьютера. Необходимо продемонстрировать на экране компьютера данные с порта контроллера, получаемые от «светофора» в виде последовательности битов	Жюри оценивает на экране компьютера данные с порта контроллера, получаемые от «светофора» в виде последовательности битов, которые соответствуют заданному жюои сигналу.	2
5.	Остановка на СТОП-линии по сигналу светофора	Оценка данного действия происходит на любом участке проезжей части, содержащей перекресток со светофором. Устройство устанавливается по	Робоавтомобиль движется по своей полосе и останавливается на «стоп-линии»	1

		центру одной полосы дорожного полотна (полоса ограничена сплошной линией справа и прерывистой слева) за 1 метр до перекрестка. Светофор горит «красным». (Сигнал светофора - «стоп»)		
6.	Пересечение перекрестка по сигналу светофора «Разрешен поворот направо».	Оценка данного действия происходит на любом участке проезжей части, содержащей перекресток со светофором, на котором возможно движение «направо». Устройство устанавливается по центру одной полосы дорожного полотна (полоса ограничена сплошной линией справа и прерывистой слева) за 1 метр до перекрестка. Сигнал светофора - «стоп» – «разрешен поворот направо».	Ехать по полосе до перекрёстка, остановиться на стоп-линии при сигнале светофора «стоп», повернуть направо по сигналу светофора «Разрешен поворот направо» и продолжить движение по полосе до следующей стоп-линии, на которой происходит остановка.	2
7.	Пересечение перекрестка по сигналу светофора «Разрешен поворот налево».	Оценка данного действия происходит на любом участке проезжей части, содержащей перекресток со светофором, на котором возможно движение «налево». Устройство устанавливается по центру одной полосы дорожного полотна (полоса ограничена сплошной линией справа и прерывистой слева) за 1 метр до перекрестка. Сигнал светофора - «стоп» – «разрешен поворот налево».	Ехать по полосе до перекрёстка, остановиться на стоп-линии при сигнале светофора «стоп», повернуть налево по сигналу светофора «Разрешен поворот налево» и продолжить движение по полосе до следующей стоп-линии, на которой происходит остановка.	3
8.	Пересечение перекрестка по сигналу светофора «Разрешено движение прямо».	Оценка данного действия происходит на любом участке проезжей части, содержащей перекресток со светофором, на котором возможно движение «прямо». Устройство	Ехать по полосе до перекрёстка, остановиться на стоп-линии при сигнале светофора «стоп», проехать прямо по сигналу светофора «Разрешено движение прямо»	3

		устанавливается по центру одной полосы дорожного полотна (полоса ограничена сплошной линией справа и прерывистой слева) за 1 метр до перекрестка. Сигнал светофора - «стоп» – «разрешено движение прямо».	и продолжить движение по полосе до следующей стоп-линии, на которой происходит остановка.	
9.	Пересечение последовательности перекрестков с разными сигналами светофора - «Разрешено движение прямо», - «Разрешен поворот налево» - «Разрешен поворот направо» в заданном порядке.	Оценка данного действия происходит на любом участке проезжей части, содержащей последовательно три перекрестка со светофорами: перекресток со светофором, на котором возможно движение «прямо», перекресток со светофором, на котором возможно движение «налево», перекресток со светофором, на котором возможно движение «направо». Устройство устанавливается по центру одной полосы дорожного полотна (полоса ограничена сплошной линией справа и прерывистой слева) за 1 метр до перекрестка. Сигнал 1-го светофора - «стоп» – «разрешено движение прямо». Сигнал 2-го светофора - «стоп» – «разрешен поворот налево». Сигнал 3-го светофора - «стоп» – «разрешен поворот направо».	Ехать по полосе до перекрёстка, остановиться на стоп-линии при сигнале светофора «стоп», проехать прямо по сигналу светофора «Разрешено движение прямо» и продолжить движение по полосе до следующей стоп-линии, на которой происходит остановка. Повернуть налево по сигналу светофора «Разрешен поворот налево» и продолжить движение по полосе до следующей стоп-линии, на которой происходит остановка. повернуть направо по сигналу светофора «Разрешен поворот направо» и продолжить движение по полосе до следующей стоп-линии, на которой происходит остановка.	4

4.2. Подтрек «MariNet»

В задаче по беспилотным плавательным устройствам необходимо собрать и запрограммировать катамаран (с дифференциальным приводом), который должен преодолеть заданный водный маршрут, используя данные УЗ-датчиков, среди препятствий и доставить груз в определенный порт в соответствии с сигналом маяка. Старт может быть начат из Точек «Старт1» и «Старт 2». При достижении порта-1 или порта -2 корабль должен

выключить оба двигателя (остановиться).

Базовый набор для конструирования:

- Комплект для сборки «корабля» с системой управления типа «катамаран» и системой крепления сервоприводов, сенсоров, системы питания и контроллера.
- Управляющая плата «Arduino», макетная плата.
- Сервоприводы, 3шт.
- Комплект датчиков для навигации по водному каналу.
- Комплект электронных компонентов.
- Комплект крепежных элементов.

Конструктор катамарана в разобранном и полностью собранном виде представлен на фотографиях:

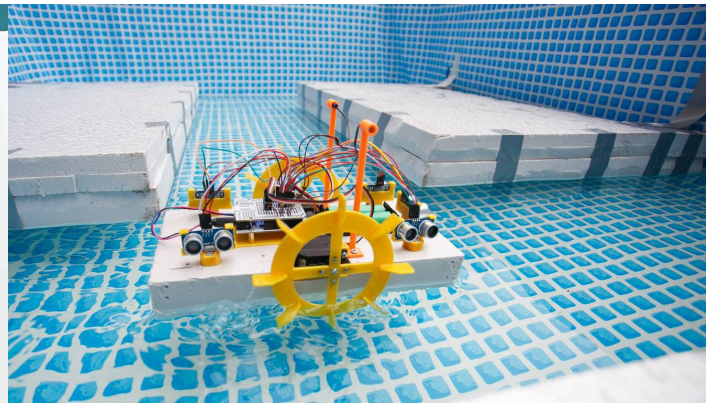
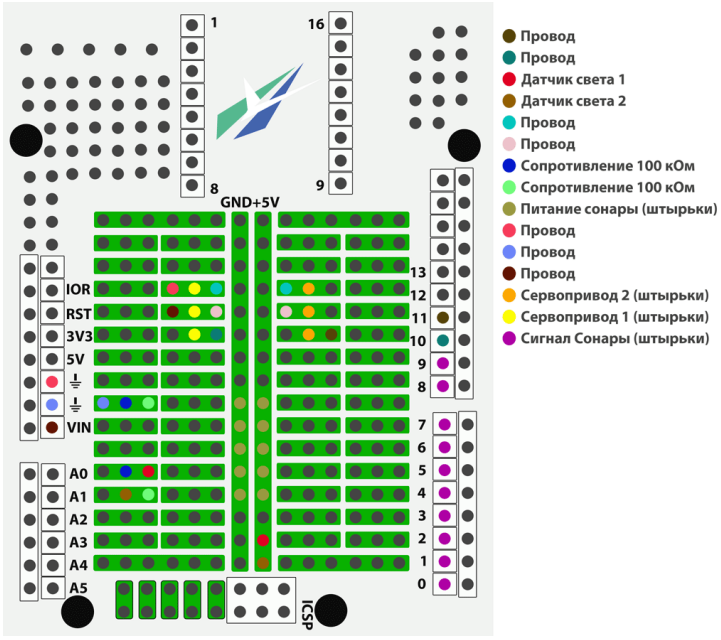
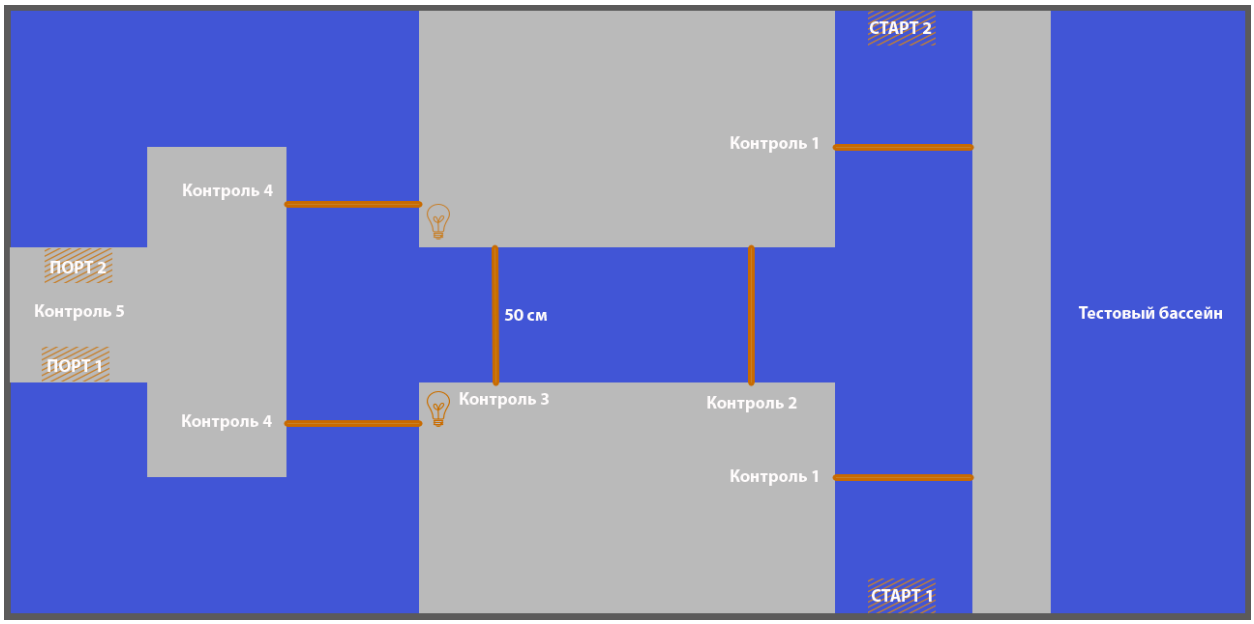


Схема монтажной платы:



Стенд (схема полигона):

Полигон представляет собой бассейн размером 4,7м на 2,2м, наполненный водой, с плавающими в нем «льдинами»:



Перечень задач, решаемых в данном подтреке:

Задача 4.2.1. Сборка роботизированного корабля с дифференциальным приводом из предоставленного конструктора. Для решения этой задачи участникам потребуются базовые навыки слесарных работ, работа с металлическим и другими видами крепежа.

Задача 4.2.2. Пайка монтажной схемы по инструкции. Для решения этой задачи участникам потребуется знание схемотехники: чтение электронной схемы, реализация её из электронных компонентов.

Задача 4.2.3. Сборка в общую схему платы Arduino, сервоприводов, датчиков света, ультразвуковых датчиков.

Задача 4.2.4. Написание программы для работы с ультразвуковыми датчиками и построения маршрута в соответствии с сигналами с них. Для решения этой задачи участникам потребуется базовое представление о программировании, знание языка C/C++, разработка алгоритмов зависимости работы сервоприводов от получения сигналов с датчиков.

Задача 4.2.5. Написание программы для выбора траектории в соответствии с сигналами маяков. Преобразование сигнала со светового датчика в команду для сервоприводов. Для решения этой задачи участникам потребуется базовое представление о программировании, знание языка C/C++.

А также набор **содержательных задач**:

- подзадача по физике: рассчитать силу трения катамарана о водную поверхность, если известно расстояние, пройденное катамараном, время, масса катамарана. Скорость катамарана считается малой;
- прикладная подзадача - расчет скорости движения катамарана в зависимости от скорости вращения ходовых двигателей;
- обработка данных сенсорной системы;
- алгоритм управления катамараном с «дифференциальным приводом» по данным сенсорной системы;
- алгоритм управления катамараном с «дифференциальным приводом» «по координатам» (расчет поворота сервомоторов);
- расчет оптимальной скорости движения беспилотника в соответствии с характеристиками серводвигателей с учетом инерции воды, для выполнения задач: движение по прямой, движение по дуге заданного радиуса, движение параллельно стене;
- определение оптимального расположения датчиков сенсорной системы. Расчет интенсивности принимаемого сенсорами сигнала, определение пороговых значений.

РЕШЕНИЕ:

Программа управления на языке С++:

```
#include <Servo.h>
#include <NewPing.h>
Servo servoLeft, servoRight;
/*порты для оборудования*/
// Кнопка старта
int Button = 0;
int ButtonPin = A2;
//передний дальнометр
const int trig = 0;
const int echo = 1;
//передний правый и задний правый дальнометры
const int FtrigR = 2;
const int FechoR = 3;
const int BtrigR = 4;
const int BechoR = 5;
//передний правый и задний правый дальнометры
const int FtrigL = 6;
```

```
18. const int FechoL = 7;
19. const int BtrigL = 8;
20. const int BechoL = 9;
21. float dist;
22. float distFL;
23. float distFR;
24. float distBL;
25. float distBR;
26. float lightL;
27. float lightR;
28. float vLeft;
29. float vRight;
30. //светодатчики левый и правый
31. const int sen0 = A1;
32. const int sen1 = A0;
33. const int v = 10; //минимальная скорость
34. const int L = -1; //константа обозначающая поворот налево
35. const int R = 1; //константа обозначающая поворот направо
36. const int N = 0; //константа обознаяающая езду прямо
37. const int E = 2;//константа обозначающая достигание финиша, выключеие
38. int k = N; //направление движения корабля, начальное значение
39. int I = 60; //расстояние от берега в см при котором нужно поворачивать
40. float turnSpeed = 90;
41. void setup()
42. {
43.   //сервопривод
44.   servoLeft.attach(11);
45.   servoRight.attach(10);
46.   /*инициализация пинов*/
47.   pinMode(trig, OUTPUT);
48.   pinMode(FtrigL, OUTPUT);
49.   pinMode(BtrigL, OUTPUT);
50.   pinMode(FtrigR, OUTPUT);
51.   pinMode(BtrigR, OUTPUT);
52.   pinMode(echo, INPUT);
53.   pinMode(FechoL, INPUT);
54.   pinMode(BechoL, INPUT);
55.   pinMode(FechoR, INPUT);
56.   pinMode(BechoR, INPUT);
57.   pinMode(sen0, INPUT);
58.   pinMode(sen1, INPUT);
```

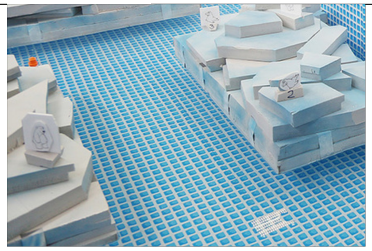
```
59.   pinMode(ButtonPin, INPUT);
60.   Serial.begin(9600);//вывод отладочной информации если она будет
61.   servoLeft.write(90);
62.   servoRight.write(90);
63. }
64. void loop()
65. {
66.   while (Button < 500) {
67.     Button = analogRead(ButtonPin);
68.     delay(10);
69.     k = N;
70.   }
71.   scan();
72.
73.   vLeft = v;
74.   vRight = v;
75.
76.   if(k == N)
77.   {
78.     if(dist > 10)
79.     {
80.       if(distFL < 10) { vLeft += 0.2 * v; }
81.       if(distFR < 10) { vRight += 0.2 * v;}
82.       if(distFR > I && distBR > I  && distFL < I){ k = R;}
83.       if(distFL > I && distBL > I  && distFR < I){ k = L;}
84.     }
85.     if(dist < 10)
86.     {
87.       if(distFL + distBL > distFR + distBR) {k = L;} else {k = R;}
88.     }
89.     //if(distFL < 20 && distBL < 20 && dist < 20 && distFR < 20 &&
distBR < 20)
90.     if(distFL < 20 && dist < 10 && distFR < 20 )
91.     {
92.       k = E;
93.     }
94.         }
95.   if(k == R)
96.   {
97.     servoLeft.write(0);
98.     servoRight.write(0);
```

```
 99.     delay(1000);
100.     servoLeft.write(95);
101.     servoRight.write(85);
102.     delay(200);
103.     setServo(vLeft*10, vRight*10);
104.     delay(1500);
105.     k = N;
106.   }
107.
108.   if(k == L)
109.   {
110.     servoLeft.write(180);
111.     servoRight.write(180);
112.     delay(1000);
113.     servoLeft.write(85);
114.     servoRight.write(95);
115.     delay(200);
116.     setServo(vLeft*10, vRight*10);
117.     delay(1500);
118.     k = N;
119.   }
120.   if(k == E)
121.   {
122.     vLeft = 0;
123.     vRight = 0;
124.     Button = 0;
125.   }
126.   setServo(vLeft,vRight);
127.   delay (100);
128. }
129. float setServo(float vLeft, float vRight)
130. {
131.   if (vLeft > 90)  { vLeft = 90; }
132.   if (vRight > 90) { vRight = 90;}
133.   servoLeft.write(90 - vLeft);
134.   servoRight.write(90 + vRight);
135.
136. }
137. float distance(int trig, int echo)//возвращает расстояние в см, на вход
138. {
139.   digitalWrite(trig, HIGH); // Подаем сигнал на выход микроконтроллера
```

```
140.   delayMicroseconds(10); // Удерживаем 10 микросекунд
141.   digitalWrite(trig, LOW); // Затем убираем
142.   unsigned time_us = pulseIn(echo, HIGH);
143.   return time_us / 58.0;
144. }
145. float scan ()
146. {
147.   dist = distance(trig, echo);
148.   delay(2);
149.   Serial.print("Front:");
150.   Serial.println(dist);
151.   Serial.println("-");
152.   distFL = distance(FtrigL, FechoL);
153.   Serial.print("L-Front:");
154.   Serial.println(distFL);
155.   delay(2);
156.   distBL = distance(BtrigL, BechoL);
157.   Serial.print("L-Back:");
158.   Serial.println(distBL);
159.   Serial.println("-");
160.   delay(2);
161.       distFR = distance(FtrigR, FechoR);
162.   Serial.print("R-Front:");
163.   Serial.println(distFR);
164.   delay(2);
165.
166.   distBR = distance(BtrigR, BechoR);
167.   Serial.print("R-Back:");
168.   Serial.println(distBR);
169.   Serial.println("-");
170.   delay(2);
171.   /*
172.   lightL = analogRead(sen0);
173.   Serial.print("L-Light:");
174.   Serial.println(lightL);
175.
176.   lightR = analogRead(sen1);
177.   Serial.print("R-Light:");
178.   Serial.println(lightR);
179.   */
180. }
```

4.2.6. Критерии оценки:

Элементы полигона и трассы:

	Название	**Описание**	
1	Акватория СТАРТА	Водное пространство, которое расположено возле противоположных бортов бассейна, обозначенные: «СТАРТ 1» и «СТАРТ 2». Размеры: 50 см х 50 см.	
2	Водный канал	Водное пространство, ограниченное с двух сторон «льдинами». Ширина канала – 50 см, длина канала - 150 см.	
3	«Маяк»	Устройство, которое расположено в конце водного канала, и в формате световой индикации передает информацию о возможности движения в двух направлениях («направо», «налево»)	
	Акватория порта разгрузки	Водное пространство, которое расположено возле одного из бортов бассейна, обозначенные: «ПОРТ 1» и «ПОРТ 2». Размеры: 50 см х 40 см. С трех сторон ограничено бортами.	
5	«Льдины»	Условные препятствия, сделаны из пенопласта, минимальная высота вертикальной поверхности льдины над поверхностью воды — 10 см. Прикрепляются к бортам, могут незначительно перемещаться в акватории бассейна — до 2 см.	

Порядок оценки и критерии:

	Действия устройства «робокатамаран», оцениваемые жюри	**Описание действия, особенности расположения на полигоне и т.д.**	**Критерий выполнения действия**	**Баллы**
1.	Выход из акватории СТАРТА	Катамаран устанавливается в любую акваторию старта (СТАРТ 1 или СТАРТ 2) в любой ориентации по отношению к борту. Катамаран должен выйти из акватории старта	Катамаран должен полностью пересечь контрольную линию «контроль 1». Действие должно быть совершено не позднее 1 минуты с момента старта.	10
2.	Выход в «водный канал»	Катамаран устанавливается в любую акваторию старта (СТАРТ 1 или СТАРТ 2) в любой ориентации по отношению к борту. Катамаран должен выйти из акватории старта, обнаружить поворот в водный канал и совершить в него поворот.	Катамаран должен полностью пересечь контрольную линию «контроль 2» . Действие должно быть совершено не позднее 5 минут с момента старта.	5
3.	Проход «водного канала»	Катамаран устанавливается в любую акваторию старта (СТАРТ 1 или СТАРТ 2) в любой ориентации по отношению к борту. Катамаран должен выйти из акватории старта, обнаружить отвод в водный канал и совершить в него поворот. Далее двигаться	Катамаран должен полностью пересечь контрольную линию «контроль 3» . Действие должно быть совершено не позднее 5 минут с момента старта. При однократном касании катамараном	5

		вдоль водного канала до выхода из него.	«льдины» начисляется штраф – 1 балл, при многократном касании штрафные баллы суммируются, но максимальный штраф не может превышать трех баллов.	
4.	Остановка в «порту разгрузки»	Катамаран устанавливается в любую акваторию старта (СТАРТ 1 или СТАРТ 2) в любой ориентации по отношению к борту. Катамаран должен выйти из акватории старта, обнаружить отвод в водный канал и совершить в него поворот. Далее двигаться вдоль водного канала до выхода из него. Определить направление движения по сигналу светофора. Дойти до Акватории порта разгрузки («ПОРТ 1» или «ПОРТ 2») и остановиться в акватории порта.	Катамаран должен полностью пересечь контрольную линию «контроль 4» и останавливается в акватории порта. Действие должно быть совершено не позднее 5 минут с момента старта.	5

4.3. Подтрек «AeroNet»

В этой задаче необходимо выполнить полетное задание, управляя квадрокоптером автономно, при помощи контроллера «Arduino» и ультразвуковых датчиков. Полетное задание: осуществить автономной взлёт, автономное движение на определенном расстоянии от поверхности, обнаружение и облет препятствия, автономная посадка.

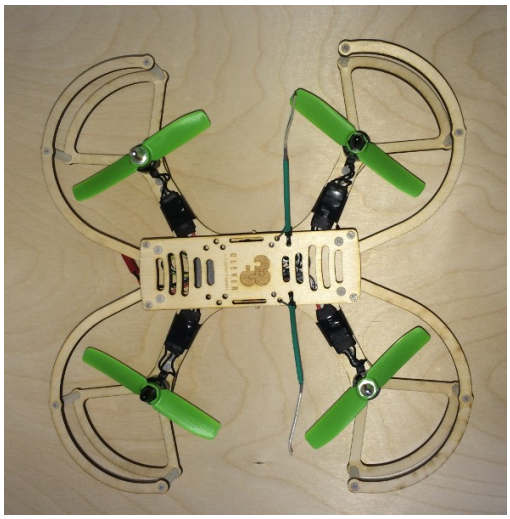
Базовый набор для конструирования (схемы, инструкции и т.д.):

- Конструктор квадрокоптера «Clever» — Квадрокоптер 280 класса (рама, моторы,

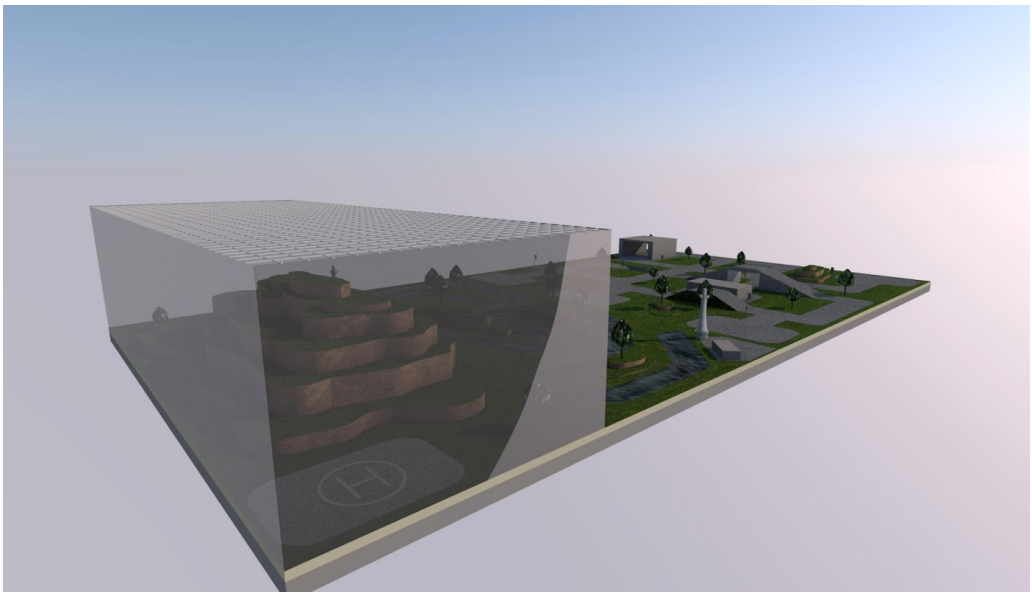
регуляторы, полетный контроллер, плата питания, пропеллеры, провода).

- Аккумулятор 2200mAh 3S 45C Lipo Pack.
- Зарядка.
- Пульт радиоуправления для безопасности FlySky FS-I6 2.4G 6CH.
- Комплект электроники (Arduino Nano, провода Dupont, JST-SH 4pin, 6 pin, коннекторы, макетки).
- УЗ-сонар XL-Maxbotix-EZ3.

Конструктор коптера в разобранном и собранном виде представлен на фотографиях:



Стенд (схема полигона) представляет собой ограниченное пространство (3 м х 6 м х 2,5 м). Стены стенда выполняются из оргстекл, сверху натянута сетка, на полу расположено препятствие. Стенд имеет следующий вид:



Для решения задач данного подтрека участники должны продемонстрировать способность воспринимать и перерабатывать большие объёмы информации, концентрироваться на поставленной задаче, разумно распределять предоставленные попытки. Задачи подтрека:

Задача 4.3.1. Заставить коптер автономно подняться и сесть, используя конструктор на основе контроллера «Arduino». Для решения этой задачи потребуются базовые навыки конструирования БВС, схемотехника, пайка, конфигурирование полётного контроллера БВС, программирование на языке C/C++.

Для решения задачи участнику необходимо правильно подключить и настроить полётный контроллер, для чего требуется иметь представление о работе основных узлов БВС и их взаимодействии. Затем следует подключить Arduino к полётному контроллеру посредством UART и написать программу, на некоторое время включающую и отключающую винты. Помимо полетного контроллера, к Arduino подключается приёмник пульта радиоуправления для того, чтобы при необходимости иметь возможность прервать полёт.

Задача 4.3.2. Используя сигналы ультразвуковых датчиков, запрограммировать коптер на взлёт, удержание высоты 50 см в течение 30 секунд со стабилизацией положения с помощью ультразвуковых сонаров и посадку. Для решения задачи потребуются: программирование на языке C/C++, цифровая обработка сигналов, основы теории автоматического управления.

Для решения задачи требуется оснастить квадрокоптер сонарами, позволяющими определять расстояния до пола и стен/препятствий. Поскольку данные с сонаров заметно

зашумлены, участникам нужно фильтровать их (например, используя скользящий медианный фильтр). Затем, на основании полученных данных, формируются управляющие команды полётному контроллеру – для этого предлагается использовать ПИД-регулятор. Таким образом, участникам предоставляется возможность применить свои знания по математике и информатике.

Задача 4.3.3. Запрограммировать коптер на взлёт, преодоление препятствия и посадку. Для решения этой задачи потребуются программирование, цифровая обработка сигналов, основы теории автоматического управления.

Требуется запрограммировать квадрокоптер на взлёт, перелёт препятствия и посадку. Для посадки в предназначенной зоне предлагается использовать сонары.

Задача 4.3.4. Запрограммировать коптер на взлёт, преодоление препятствия, посадку, взлёт, преодоление препятствия и посадку на стартовой позиции. Для решения этой задачи потребуются программирование, цифровая обработка сигналов, основы теории автоматического управления. Это усложненная версия предыдущего задания, включает в себя возврат в зону старта.

Также потребуется решение следующих **содержательных задач**:

- Расчет подъемной силы для осуществления вертикального взлета.
- Расчет параметров двигателей для стабилизации коптера в воздухе.
- Расчет параметров полета для осуществления посадки в заданную точку.
- Обработка данных сенсорной системы.
- Составление алгоритма управления полетными двигателями на основании данных сенсорной системы.

РЕШЕНИЕ:

```
#include "msppg.h"
#include <inttypes.h>
#include <PID_v1.h>

const int RC_INTERVAL = 500;

int trigPin = 9;
int echoPin = 8;
int trigPinFront = 7;
int echoPinFront = 6;
int D0 = 11;
int D1 = 12;
```

```
13.
14. int midRoll = 1360;
15. int midPitch = 1555 + 70;
16.
17. int d0 = 0;
18. int d1 = 0;
19. int i = 1550;
20. int duration = 0;
21. double cm = 0;
22. int s = 0;
23. double h = 70;
24. int delta = 0;
25.
26. int maxthr = 1650;
27. int minthr = 1500;
28. int midthr = 1600;
29. int step_up = 2;
30. int step_down = 1;
31. int counter = 0;
32.
33. double out = 0;
34. double Kp = 1.5, Ki = 3.7 , Kd = 1;
35.
36. PID myPID(&cm, &out, &h, Kp, Ki, Kd, DIRECT);
37.
38. void setup()
39. {
40.   delay(5000);
41. // serial config
42.   Serial.begin(115200);
43.   pinMode(trigPin, OUTPUT);
44.   pinMode(echoPin, INPUT);
45.   pinMode(trigPinFront, OUTPUT);
46.   pinMode(echoPinFront, INPUT);
47.   pinMode(D0, INPUT);
48.   pinMode(D1, INPUT);
49.
50.   myPID.SetMode(AUTOMATIC);
51.   myPID.SetOutputLimits(-100,200);
52.
53.   write_rc_command(1500, 1500, 1500, 1000, 0);
```

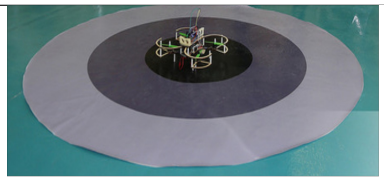
```
54.   delay(200);
55.
56.   write_rc_command(1500, 1500, 1500, 1000, 1);
57.   delay(200);
58.
59.   //write_rc_command(1500, 1500, 1500, 1650, 1);
60.   //delay(1000);
61.  }
62. //---------------------------------------------------------------
63. void loop()
64. {
65.
66.   //write_rc_command(1500, 1500, 1500, 1300, 1);
67.   //delay(100);
68.
69.   // write_debug("debug_msg");
70.
71.   //  write_attitude_request();
72.   //  write_rc_request();
73.   d0 = digitalRead(D0);
74.   d1 = digitalRead(D1);
75.
76.   Serial.print(" d0 = ");
77.   Serial.print(d0);
78.   Serial.print("; d1 = ");
79.   Serial.print(d1);
80.
81.   digitalWrite(trigPin, LOW);
82.   delayMicroseconds(2);
83.   digitalWrite(trigPin, HIGH);
84.   delayMicroseconds(10);
85.   digitalWrite(trigPin, LOW);
86.   duration = pulseIn(echoPin, HIGH);
87.   cm = duration / 58;
88.
89.
90.   delay(50);
91.   Serial.print("; cm = ");
92.   Serial.print(cm);
93.   // delay(100);
94.   if ((cm > 1) && (cm<150)) {
```

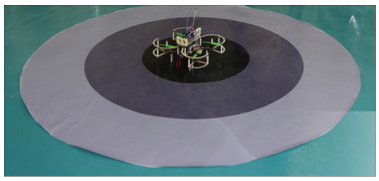
```
95.     // delta = h - cm;
96.
97.     Serial.print("; out = ");
98.     Serial.print(out);
99.
100.     myPID.Compute();
101.    i = out + midthr-30;
102.  }
103.  else {
104.    i = midthr;
105.  }
106.
107.  digitalWrite(trigPinFront, LOW);
108.  delayMicroseconds(2);
109.  digitalWrite(trigPinFront, HIGH);
110.  delayMicroseconds(10);
111.  digitalWrite(trigPinFront, LOW);
112.  duration = pulseIn(echoPinFront, HIGH);
113.  s = duration / 58;
114.
115.  Serial.print("; s = ");
116.  Serial.print(s);
117.  Serial.print("; counter = ");
118.  Serial.print(counter);
119.
120.  if ((counter>100)||((counter>100)&&(s < 200)&&(s > 170))) {
121.    for (i; i > 1300; i -= 10) {
122.      write_rc_command(1500, 1450, midRoll, i, 1);
123.      delay(100);
124.    }
125.    write_rc_command(1500, 1500, 1500, 1000, 1);
126.    delay(100);
127.    write_rc_command(1500, 1500, 1500, 1000, 0);
128.    delay(100);
129.    while (1) i = 0;
130.  }
131.
132.  if (d1 == 1) {
133.    write_rc_command(1500, 1500, 1500, 1450, 1);
134.    delay(5000);
135.  }
```

```
136.
137.   if (d0 == 1) {
138.     write_rc_command(1500, 1500, 1500, 1000, 1);
139.     delay(5000);
140.   }
141.   else {
142.     Serial.print("; i = ");
143.     Serial.println(i);
144.     write_rc_command(1500, midPitch, midRoll, i, 1);
145.     delay(15);
146.   }
147.   // read all data from serial
148.   /*while (Serial.available() > 0)
149.   {
150.     state::parser.parse(Serial.read());
151.   }*/
152.   counter += 1;
153. }
154.
155. //-----------------------------------------------------------------
156. int ultrasonic_measure(int echo_pin, int trig_pin)
157. {
158.   pinMode(trig_pin, OUTPUT);
159.   digitalWrite(trig_pin, LOW);
160.   delayMicroseconds(2);
161.   digitalWrite(trig_pin, HIGH);
162.   delayMicroseconds(10);
163.   digitalWrite(trig_pin, LOW);
164.   pinMode(echo_pin, INPUT);
165.   return pulseIn(echo_pin, HIGH);
166. }
```

4.3.5. Критерии оценки:

Элементы полигона и трассы:

	Название	Описание	
1	Зона СТАРТА	Представляет собой три вложенных круга: внутренний круг – радиус 40 см, средний круг – радиус 70 см, большой круг – 100 см. Центр Зоны	

		СТАРТА расположен посередине первой полетной зоны полетного куба (перед препятствием)	
2	Препятствие	Разделяет первую и вторую зоны полетного куба пополам, состоит из плоских зданий высота по всей длине не превышает 50 см.	
3	Зона ФИНИША	Представляет собой три вложенных круга: внутренний круг – радиус 40 см, средний круг – радиус 70 см, большой круг – 100 см. Центр Зоны СТАРТА расположен посередине второй полетной зоны полетного куба (перед препятствием)	

Порядок оценки и критерии:

	Действия устройства «коптер», оцениваемые жюри	**Описание действия, особенности расположения на полигоне и т.д.**	**Критерий выполнения действия**	**Баллы**
1.	Взлет-посадка	Коптер устанавливается в центр зоны старта в любой ориентации по отношению к препятствию. Коптер должен подняться в воздух и приземлиться.	Коптер поднялся в воздух и приземлился	7
2.	Стабилизация на высоте	Коптер устанавливается участником в центр зоны старта самостоятельно. Коптер должен подняться в воздух, зафиксироваться на высоте 50 см и производить удержание высоты 50 см в течение: После этого коптер должен совершить посадку посадку	Коптер поднялся в воздух, зафиксировался на высоте 50 см и приземлился. Количество баллов определяется длительностью фазы стабилизации: - 5 с — 3 балла - 10 с – 4 балла - 20 с – 5 баллов - 30 с – 6 баллов	3-6
3.	Облет	Коптер устанавливается	Коптер поднялся в	7

	препятствия	участником в центр зоны старта самостоятельно. Коптер должен подняться в воздух, перелететь через препятствие и приземлиться. При этом не касаться стен и потолка во время выполнения полёта	воздух, обнаружил препятствие, перелетел препятствие и приземлился. При однократном касании коптером препятствия начисляется штраф – 1 балл, при многократном касании штрафные баллы суммируются, но максимальный штраф не может превышать трех баллов.	
4.	Посадка в зоне Финиша	Коптер устанавливается участником в центр зоны старта самостоятельно. Коптер должен подняться в воздух, перелететь через препятствие и совершить посадку в обозначенную кругом зону ФИНИША Подзадачи: не касаться стен и потолка во время выполнения полёта штраф за касание (1 балл, но не более двух штрафных баллов)	Коптер поднялся в воздух, обнаружил препятствие, перелетел препятствие и приземлился в Зоне ФИНИША. Количество баллов зависит от точности приземления: - красная зона (малый круг) -5 баллов - зеленая зона (средний круг) – 4 балла - синяя зона (большой круг) – 3 балла Измерения производятся по проекции коптера на горизонтальную поверхность.	3-5

4.4. Подтрек «SpaceNet»

Необходимо собрать спутник связи и наладить радиосвязь с базой, чтобы по командам с земли спутник выполнял следующие задачи:

1. Мигать светодиодом с частотой 2 раза в секунду по команде start.
2. Пересылать на землю название команды по команде hello.
3. Включать звуковой сигнал по команде с земли beep.
4. Орентировать спутник в пространстве для наведения лазера на координатную плоскость по команде с земли aim**.**

Базовый набор для конструирования (схемы, инструкции и т.д.):

- Пластиковый корпус.
- Контроллер на базе Arduino (Spaceuino CPU).

- Плата питания (Spaceuino Power) + батарейки.
- Плата полезной нагрузки (Spaceuino BreadBoard).
- Набор датчиков, светодиоды, лазерная указка.
- Крепеж.

Конструктор спутника в разобранном и собранном виде представлен на фотографиях:

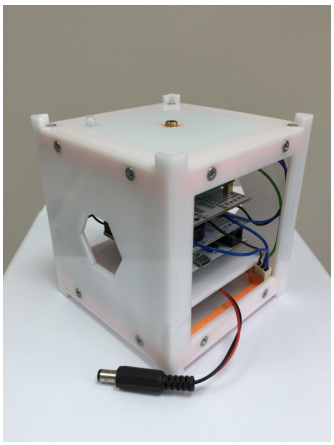
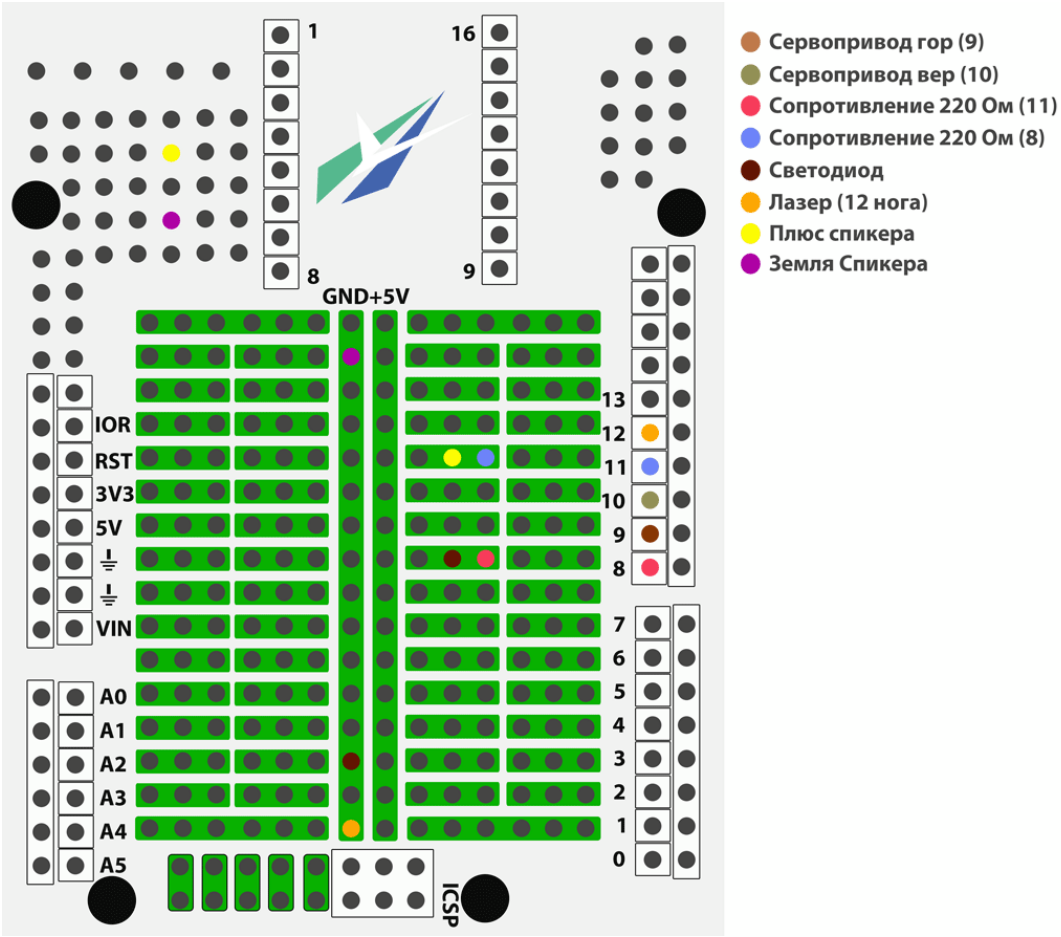
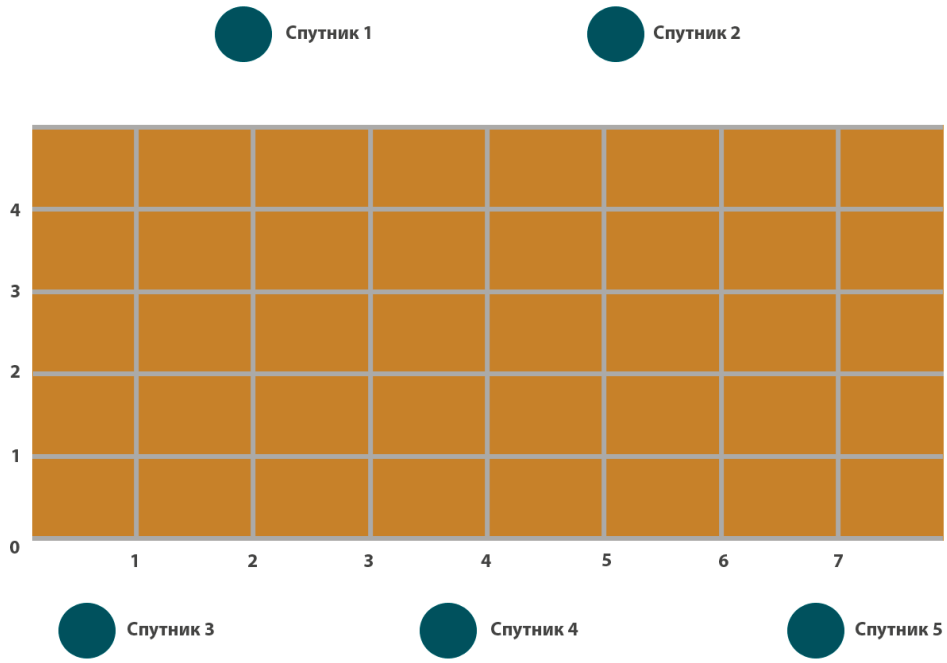


Схема макетной платы:



Стенд (схема полигона):



Данный подтрек подразумевает решение следующих задач:

Задача 4.4.1. Собрать макет спутника связи из предоставленного конструктора и разместить его на специальной стойке. Для решения этой задачи участникам потребуются: базовые навыки слесарных работ, пайка, схемотехника.

Задача 4.4.2. Программирование спутника. Налаживание радиосвязи со спутником: отправка данных и получение ответов на них, включение светодиодов и спикеров на спутнике по запросу. Для решения этой задачи участникам потребуются базовые навыки программирования, знание языка C++.

Задача 4.4.3. Калибровка сервоприводов, управляющих движением макета спутника. Указание лазером на координатные точки реальной поверхности по команде с базы. Для решения этой задачи участникам потребуются базовые навыки программирования, стереометрия.

Решение включает в себя подзадача по геометрии: расчитать углы поворота спутника (вращающих серводвигателей), зная высоту подвеса h, расстояние подвеса до ближайшей точки по оси X, если шаг координатной сетки равен одному метру. Размер координатной плоскости — 7x7 метров.

РЕШЕНИЕ:

Программа управления на языке C++:

```
#include <VarSpeedServo.h>
#include <math.h>

#define _IDLE 0
#define _ALIVE 1
#define _AIM 2
int state = _IDLE;
bool speak = 0;
long x = 0, y = 0;
unsigned long timer, speak_time, rotate_time;
bool diode = 0;
const double h = 212.5;

long rp = 500, ry = 500;
VarSpeedServo pitch, yaw;
void setup() {
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(11, OUTPUT);
  Serial1.begin(4800);
  pitch.attach(9, 350, 2090);
  yaw.attach(10, 680, 2380);
  digitalWrite(2, HIGH);
  digitalWrite(3, HIGH);
}
void loop(){
  while(Serial1.available() > 0){
    String s = Serial1.readStringUntil('\n');
    if(s == "start"){
      state = _ALIVE;
      timer = millis() + 250;
      break;
    }
    if(s == "Hello"){
      Serial1.write("%teamname%");
      break;
    }

    if(s == "Beep"){
```

```
42.       speak = 1;
43.       speak_time = millis() + 2000;
44.       //tone(11, 1500, 200);
45.       Serial1.write("Beeping");
46.       break;
47.     }
48.     if(s.startsWith("Aim")){
49.       y = s.substring(4,5).toInt();
50.       x = s.substring(6).toInt();
51.       x = 7 - x;
52.       y = 7 - y;
53.       x *= 100;
54.       y *= 100;
55.       //x += 0;
56.       y += 5;
57.       state = _AIM;
58.       digitalWrite(12, HIGH);
59.       double q = sqrt(x*x + y*y);
60.       long anglep = 180*(atan(q/h)/(M_PI)),
61.       angley = 180*(atan((double)x/y)/(M_PI));
62. //      pitch.write(180 - anglep);
63. //      yaw.write(90 + angley);
64.       pitch.write(180, 20, 1);
65.       yaw.write(90, 20, 1);
66.       pitch.write(180 - anglep, 20, 1);
67.       yaw.write(90 + angley, 20, 1);
68.       Serial1.write("Aiming to x = ");
69.       Serial1.print(x);
70.       Serial1.write(", y = ");
71.       Serial1.print(y);
72.       Serial1.write(", pitch = ");
73.       Serial1.print(anglep);
74.       Serial1.write(", yaw = ");
75.       Serial1.print(angley);
76.       break;
77.     }
78.     if(s == "Angle"){
79.       x = Serial1.parseInt();
80.       y = Serial1.parseInt();
81.       digitalWrite(12, HIGH);
82.       pitch.write(x);
```

```
      yaw.write(y);
      Serial1.write("Aiming to x = ");
      Serial1.print(x);
      Serial1.write(", y = ");
      Serial1.print(y);
      break;
    }
    if(s == "test"){
      rotate_time = millis() + 2000;
      yaw.write(180);
      Serial1.write("Rotating");
      break;
    }
    if(s == "rp"){
      long qw = Serial1.parseInt();
      rp += qw;
      pitch.writeMicroseconds(rp);
      Serial1.write("Current pitch: ");
      Serial1.print(rp);
      break;
    }
    if(s == "ry"){
      long qw = Serial1.parseInt();
      ry += qw;
      yaw.writeMicroseconds(ry);
      Serial1.write("Current yaw:");
      Serial1.print(ry);
      break;
    }
  }

  if(state == _ALIVE){
    if(millis() > timer){
      diode = !diode;
      digitalWrite(6, diode ? HIGH : LOW);
      timer = millis() + 250;
    }
  }
  if(speak){
    if(millis() > speak_time){
      //tone(11, 1500, 200);
```

```
124.         speak_time = millis() + 2000;
125.       }
126.     }
127. }
```

4.4.4. Критерии оценки:

Элементы полигона и трассы:

	Название	Описание	
1	Стойка подвеса «спутника»	Стойки подвеса спутника располагаются по периметру полигона АВТОНЕТ. Высота стоек варьируется от 200 до 250 см. Стойки снабжены сервоприводами, которые управляются со спутника. За каждой командой определяется стойка подвеса в соответствии со жребием (случайным образом).	
2	Базовая станция	Радиомодуль жюри, при помощи которого, осуществляется и проверяется радиосообщение со спутником	
3	Объект инфраструктуры	Место на полигоне с заданными координатами, отмеченное специальной меткой. Начало координат, координаты точек объектов инфраструктуры (10 штук) жюри наносит на полигон перед началом практической части	

Порядок оценки и критерии:

	Действия устройства «спутник», оцениваемые судейской бригадой	**Описание действия, особенности расположения на полигоне и т.д.**	**Критерий выполнения действия**	**Баллы**
1.	«Запуск» спутника	Участник устанавливает спутник на стойку, включает питание, световая индикация не работает. Далее происходит проверка работоспособности	Спутник начинает мигать светодиодом с частотой около двух раз в секунду после команды «start», посланной	5

		спутника «с Земли». Для этого посылается команда с базовой станции «start», спутник должен начать мигать светодиодом с частотой около двух раз в секунду.	представителем жюри с базовой станции	
2.	Получение ответа от спутника в виде радиосигнала	Посылается команда «Hello» с базовой станции. В ответ на команду «Hello» от базовой станции отправить по радиоканалу название команды (не прекращая предыдущие действия).	На базовой станции фиксируется ответ в виде текста «название команды»	5
3.	Получение ответа от спутника в виде световой и звуковой индикации	Посылается команда «Beep» с базовой станции. В ответ на команду «Beep» - начинает мигать светодиодом и издавать звук на частоте 1.5 кГц каждые 2 секунды.	В ответ на команду базовой станции «Beep» - спутник начинает мигать светодиодом и издавать звук на частоте 1.5 кГц каждые 2 секунды.	5
4.	Лазерная подсветка объекта инфраструктуры	Посылается команда «Aim:X,Y» с базовой станции. В ответ на команду «Aim:X,Y» - навести лазерную указку на объект инфраструктуры. Измерение производится по трем точкам (разноудаленных от стойки подвеса спутника), результат - среднее арифметическое трех попыток, округленное до целого	Жюри последовательно определяет три объекта инфраструктуры (три точки на полигоне). Спутник наводится лазерной указкой. Световая точка проецируется на полигон. Точность попадания определяется линейными измерениями, баллы начисляются в соответствии с: Попадание в окрестность радиуса: • 10 см - 10 баллов • 25 см - 6 баллов • 50 см - 3 балла	3-10