§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.

Базовый набор для конструирования:

- Набор для конструирования беспилотного автомобиля
- Набор для конструирования беспилотного корабля
- Набор для конструирования квадрокоптера
- Набор для конструирования бортового компьютера спутника навигации

Инструментарий:

- Ноутбук, предоставляемый организаторами один на команду – 2 штуки. Можно использовать свои ноутбуки в неограниченных количествах
- Паяльная станция – 1 на команду;
- Элементы питания
- Набор ручного инструмента (бокорезы, пинцет, кусачки, длинногубцы, макетный нож, набор отверток)
- Комплект расходных материалов (флюс, припой, изолента, пенопласт)

Программное обеспечение:

- Среда программирования Arduino IDE
- Браузер
- Acrobat Reader (или аналог)
- LibrePilot 16.09
- OpenPilot-RELEASE 15.02

Командная часть заключительного этапа имеет продолжительность 4 дня (всего 23 астрономических часа), 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 (плюс общий проезд).

Подведение итогов:

Итоговый результат определяется суммированием лучших попыток в каждом из четырех подтреков и общей задачи. В каждом подтреке можно заработать до 50 баллов (уточнение в Приложении.) Максимальное количество баллов - 250. 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.

По итогам командного этапа выявляется 1 команда – победитель (на базе наивысшего итогового балла по итогам зачетов на всех подтреках, независимо от возраста и учебного класса участников).

Тренировки:

Подход к стенду свободный, если нет других желающих. Иначе каждой команде даётся 10 минут на тестирование.

Общие правила:

Для всех задач используются общие дополнительные правила работы на полигоне.

Требования:

Все модели начинают выполнение (сдачу) задания после нажатия кнопки старт. Кнопку нажимает преподаватель. Модель может быть установлена в зоне старта под углом, ставит модель на трек преподаватель. В случае спутника - подключение производить преподаватель. Модель не должна рассыпаться в руках (как критерий: выдерживать падение с высоты 5 см на стол полигона). Указания светофоров и маяков разыгрываются жеребьевкой перед стартом прохождения трека.

Штрафы:

№	Штрафы	Баллы
1	Разрядка аккумулятора во время сдачи задачи включая спутник	Незачет попытки
2	Запуск коптера вне полигона и без преподавателя	10 баллов
3	Заход в зону преподавателей без разрешения преподавателя	5 баллов
4	Устройство не начало выполнение задания после нажатия преподавателем кнопки старта	Незачет попытки

4.1 Подтрек «AutoNet»

Задача:

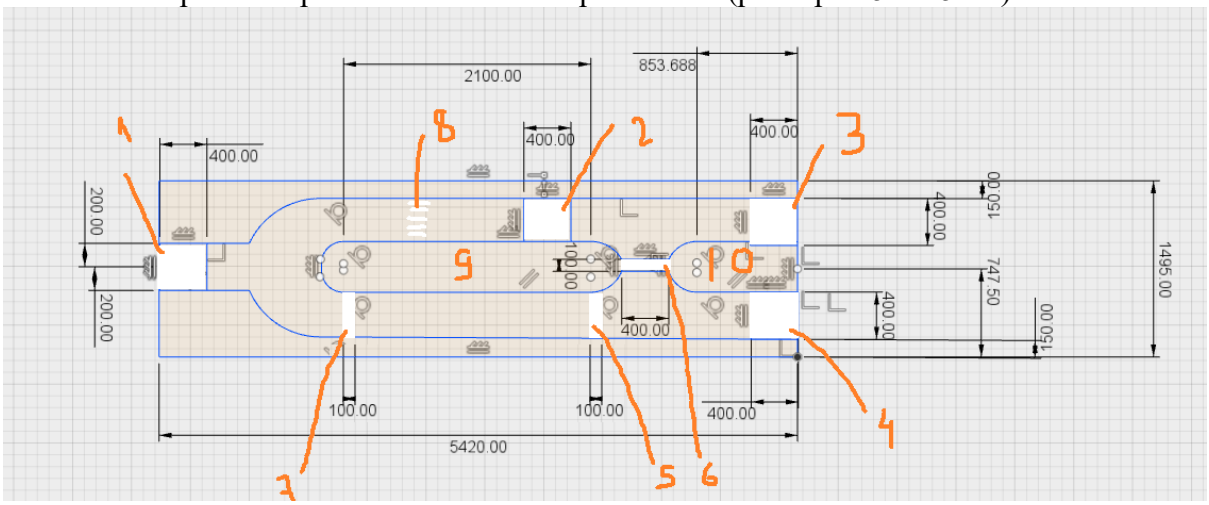
В задаче по беспилотным автомобилям нужно будет разместить электронику и сенсоры на четырехколесном шасси (с автомобильной кинематикой) для решения задачи автономного доставка груза по городу из 1 из 2 портов отгрузки в аэропорт, ориентируясь по дорожной разметке, элементам инфраструктуры и сигналам ИК-светофора[1].

Базовый набор:

- Шасси с мотором, регулятором хода и рулевым сервоприводом
- Зарядное устройство
- Управляющая плата Arduino
- Набор датчиков
- Комплект крепежных элементов
- Навигационная метка
- Радиомодуль
- Набор для сборки ИК приемника и стабилизатора
- Пьезодинамик

Схема полигона:

Баннерное покрытие с нанесенной разметкой (размер: 1.5м х 5.4м).



Абсолютно матовый баннер (литое полотно 510гр, разрешение печати 720 dpi) или фототкань 1495*5420 мм плюс люверсы по краям (не даны на чертеже). Все внутренние линии (на чертеже – синие. за исключением границ) – белого цвета толщиной 25 мм. Дорожное полотно ограничено внутренними линиями (на чертеже – синие), за исключением элементов ландшафта 9 и 10 – это абсолютно черная поверхность без зернистости или оттенков серого.

Элементы разметки:

1 – финальный бокс прибытия;

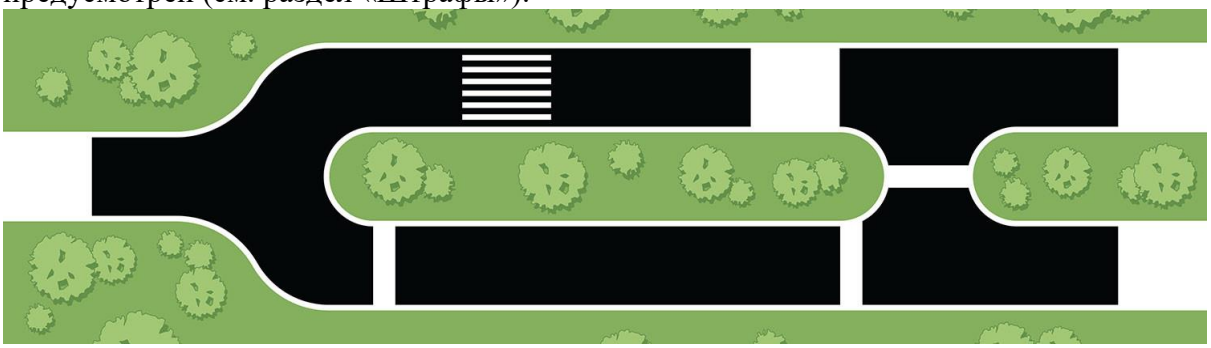
2 – «зона досмотра», 1-ый чекпоинт правой ветки (см. далее);

3,4 – две зоны старта, определяются судьей, для каждой из них справа расположен ИК-светофор (значения сигналов: «направо» или «налево»);

5 – элемент разметки, напротив которой справа стоит ИК-светофор (значения сигналов:

[1] Устройство, которое расположено на указанном месте трассы, и в формате ИК-сигнала и световой индикации передает информацию о возможности движения в трех направлениях («направо», «прямо», «налево») либо «остановки».

«остановка» или «прямо»), 1 чекпоинт правой ветки (см. далее);
6 – не задействован в задачах, разметка используется по желанию участников как вспомогательная;
7 – 2-ой чекпоинт правой ветки (см. далее);
8 – разметка пешеходного перехода, судья может поставить на нее модель пешехода, занимающую не менее 2/3 ширины дорожного покрытия, 2-ой чекпоинт правой ветки;
9,10 – «жилой квартал» полигона с расположенными на нем домами, заезд ТС не предусмотрен (см. раздел «Штрафы»).



Баллы:

	Действия устройства, оцениваемые жюри	Задача	Баллы
1.	Сборка работоспособного устройства с пайкой	Машина полностью собрана: все датчики электрически подключены правильно (хотя бы в одном экземпляре) (датчик линии, сервопривод - обязательно должен быть подключен через стабилизатор напряжения (см. Рисунок 1), уз-сонар, пищалка, светодиодная метка) - и может ехать вперед 5 секунд и назад 5 секунд.	10
3.	УЗ	Машинка должна следовать вперёд и назад за препятствием на дистанции 30 см.	2
4.	Пайка ИК-приемника	Собрать ИК-приемник (см. Рисунок 2), вывести на экран получаемую с помощью ИК-приемника битовую последовательность	4
5.	Езда по левой ветке трека	Машина стартует перед первым чекпоинтом, пересекая белую линию начинает мигать светодиодной меткой, по пересечении второй линии датчиком линии - перестает мигать и единожды бибикает.(Задание на пайку и чтение datasheet транзистора) ВНИМАНИЕ: машинка должна ехать без сопровождения спутника, чтобы мигающая метка не влияла на качество обратной связи от ПО камеры.	4
6.	Задание на радио	Вывести свои стартовые координаты, полученные по	2

		радио со спутника, в последовательный порт.	
7.	Упрощённый проезд по треку без спутника	Засчитывается прохождение верных чекпойнтов без спутника, по 2 балла за первый и второй чекпойнт, 2 балла за финиш.	8
8.	Комплексный проезд по треку*	Машина начинает движение из одного из стартовых боксов . Получает свою текущую координату от спутника. Получает сигнал ИК-светофора, который задаёт желаемую траекторию движения из 4 вариантов. Засчитывается прохождение верных чекпойнтов, по 3 балла за первый и второй чекпойнт, 4 балла за финиш.	20

ШТРАФ за разрушение дома - 2 балла

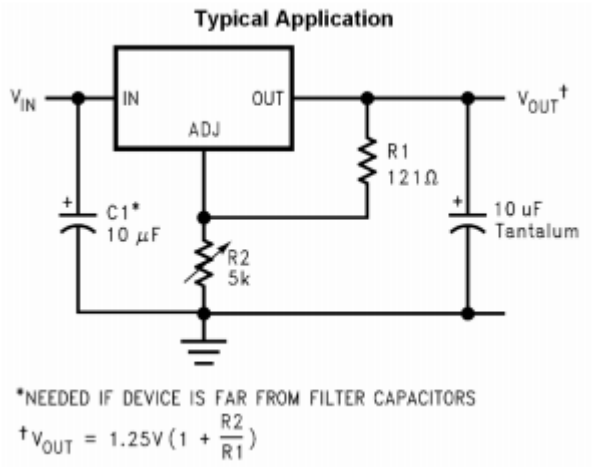


Рисунок 1 – Схема для пайки стабилизатора

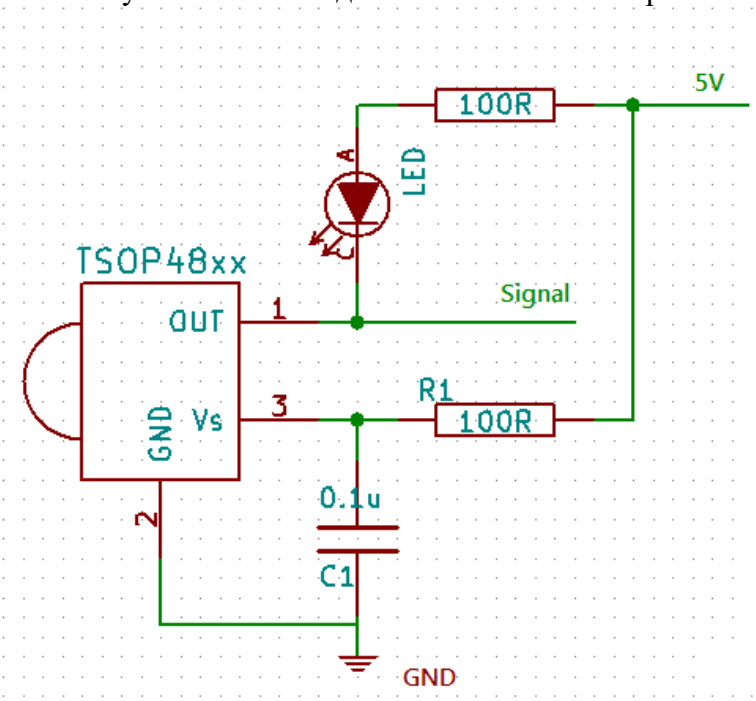


Рисунок 2 – Схема для сборки ИК-приемника

Описание чекпойнтов:

Правая ветка:

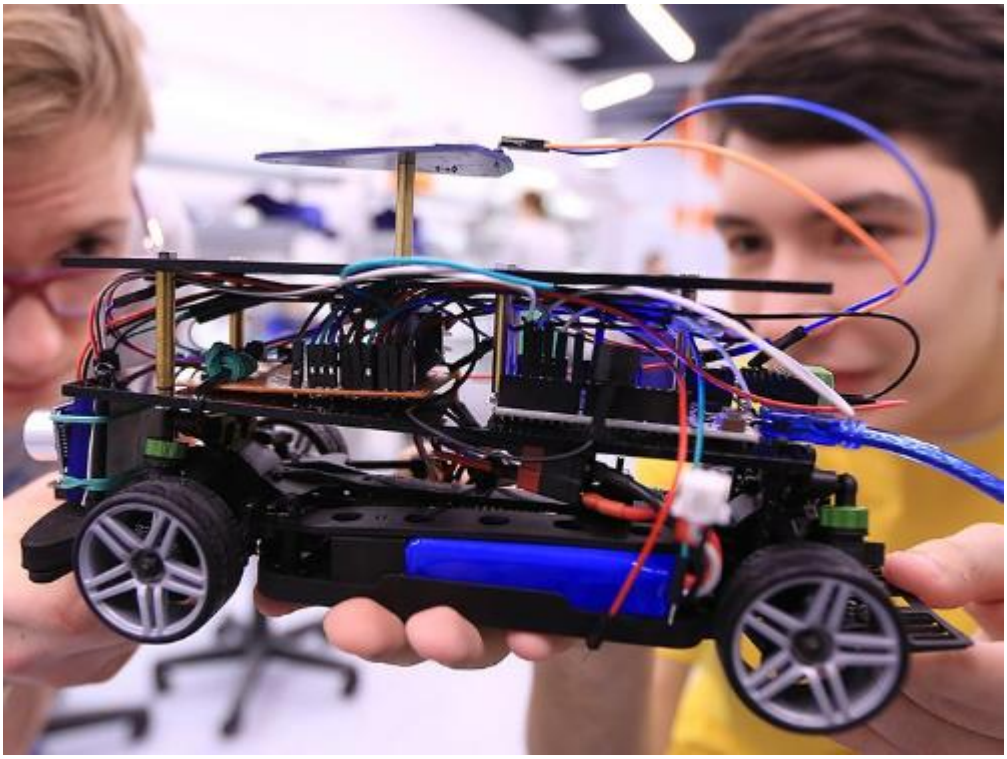
1) Заехав датчиком линии спереди и позади корпуса машины, последняя останавливается и фиксируется на 10 секунд. Продолжает движение по истечении указанного времени.
2) На пешеходном переходе (в начале или конце - попеременно) стоит пешеход, занимающий не менее ⅔ ширины полосы. Задача машины - остановиться за 20 см от пешехода, прореагировав при помощи сонаров (а не привязано к координатам).

Левая ветка:

1) Напротив первого чекпоинта стоит светофор, на котором горит красный сигнал. При проезде машинка останавливается, пока не загорится разрешающий (зеленый прямо) сигнал светофора. Машинка продолжает движение по треку.
2) Заехав передним датчиком линии (бампером), машинка останавливается на 5 секунд и единожды сигналит пьезодинамиком.

В любых проездах разрешается выезжать одним колесом за полосу.

Вариант сборки устройства:



Полное решение:

```
#include <Servo.h>
#include <stdint.h>
#include "irreceiver.h"
#include <string.h>
#include <NewPing.h>

#define TRIGGER_PIN  5  // Arduino pin tied to trigger pin on the
ultrasonic sensor.
#define ECHO_PIN     6  // Arduino pin tied to echo pin on the
ultrasonic sensor.
#define MAX_DISTANCE 200 // Maximum distance we want to ping for (in
centimeters). Maximum sensor distance is rated at 400-500cm.
```

```
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); // NewPing setup
of pins and maximum distance.

#define SetPin 2

const uint8_t IR_RECEIVER_PIN = 7;
IrReceiver receiver(IR_RECEIVER_PIN);
const uint8_t SIGNAL_STOP = 0xf0;
const uint8_t SIGNAL_FORWARD = 0xc0;
const uint8_t SIGNAL_LEFT = 0xa0;
const uint8_t SIGNAL_RIGHT = 0x90;

String str;
int x, y, angle;
int StartX, StartY;
int SatLoc;
int spd = 90;
Servo myservo;
Servo Motor;

#define Left 125
#define Mid 90
#define Right 55
// 90 центр
// 125 лево
// 55 право


void setup()
{ pinMode(SetPin, OUTPUT);
  digitalWrite(SetPin, LOW);
  Serial.begin(9600);
  Serial1.begin(9600);
  myservo.attach(9);
  myservo.write(90);
  Motor.attach(8);
  Motor.write(90);
  receiver.init();
  ChannelDebag();

  byte flag = 1;
  while (flag)
  { if (Serial1.available())
    { spd = Serial1.parseInt();
      Serial.print("NewSpeed");
      str = Serial1.readStringUntil('\n');// LF
      if (spd == -3)
      {
        flag = 0;
      }
    }
    Serial.println(spd);
    Motor.write(spd);//////////////////////////////////////
    //myservo.write(spd);
  }
  Motor.write(90);
  ChannelCoor();
  MoveSat(-300);
```

```
  MoveSat(78);
  GetCoordinate();
  StartX = x;
  StartY = y;
  spd=97;
}

uint8_t signal = SIGNAL_FORWARD;
int distance = 0;
void loop()
{ GetCoordinate();
//  if (Serial1.available())
//  { spd = Serial1.parseInt();
//    str = Serial1.readStringUntil('\n');// LF
//    Serial.print("NewSpeed");
//    Serial.println(spd);
//  }

  if (receiver.isDataReceived())
  {
    signal = receiver.getReceivedData() >> 4;
    Serial.print("Incoming signal: ");
    Serial.print(signal, HEX);
    Serial.println("!");
  }

  distance = sonar.ping_cm();
  Serial.print("Distance: ");
  Serial.println(distance);
  if ((distance < 30) and (distance > 8))
  { Serial.println("Barrier");
    Motor.write(90);
    digitalWrite(13, HIGH);
  }
  else
  {
    if (signal == SIGNAL_FORWARD)
    { Serial.print("No problem -<< ");
      Serial.print(spd);
      Serial.println(" >>");
      if (y < 154)
      { Motor.write(90);
        if (SatLoc > 140)
        { while (1)
          { /////////END END END END/////////////}
            digitalWrite(13, HIGH);
            delay(500);
            digitalWrite(13, LOW);
            delay(500);
          }
        }
        MoveSat(18);
      }
      else
      { if (x > StartX)
        { myservo.write(Mid + 10);
          Serial.println("NaLevo");
```

```
        }
        else
        { myservo.write(Mid - 10);
          Serial.println("NaPravo");
        }
        Motor.write(spd);
      }
      digitalWrite(13, LOW);
    }
    if (signal == SIGNAL_STOP)
    { Serial.println("Stop");
      Motor.write(90);
      digitalWrite(13, HIGH);
    }
  }
}


void MoveSat(int steps)
{ ChannelSatelite();
  Serial.print("MoveSat| ");
  Serial.println(steps);
  Serial1.println(steps);
  while (1)
  { if (Serial1.available() > 0)
    {
      str = Serial1.readStringUntil('\n'); //have /r/n sting
      Serial.println(str);
      if (str.substring(0,4) == "STOP")
      { SatLoc += steps;
        ChannelCoor();
        return ;
      }
      if (str.substring(0,4) == "END_")
      { SatLoc = 0;
        ChannelCoor();
        return ;
      }
    }
  }
}


void GetCoordinate()
{
  while (1)
  {
    if (Serial1.available() > 0) {
      str = Serial1.readStringUntil('\n');
      //str = "00;1019;0609;192";
      Serial.println(str);
      x    = str.substring(3, 7).toInt();
      y    = str.substring(8, 12).toInt();
      angle = str.substring(13, 16).toInt();
      return ;
    }
  }
}
```

```
void ChannelSatelite()
{ digitalWrite(SetPin, LOW);
  delay(200);
  Serial.println("Start HC-12 Config");
  Serial1.println ("AT+C010");
  delay(200);
  while (Serial1.available() > 0)
  { str = Serial1.readStringUntil('\n');// LF
    Serial.print ("Back: ");
    Serial.println (str);
  }
  digitalWrite(SetPin, HIGH);
  delay(100);
  Serial.println("Save & Exit");
}

void ChannelCoor()
{ digitalWrite(SetPin, LOW);
  delay(200);
  Serial.println("Start HC-12 Config");
  Serial1.println ("AT+C001");
  delay(200);
  while (Serial1.available() > 0)
  { str = Serial1.readStringUntil('\n');// LF
    Serial.print ("Back: ");
    Serial.println (str);
  }
  digitalWrite(SetPin, HIGH);
  delay(100);
  Serial.println("Save & Exit");
}

void ChannelDebag()
{ digitalWrite(SetPin, LOW);
  delay(200);
  Serial.println("Start HC-12 Config");
  Serial1.println ("AT+C003");
  delay(200);
  while (Serial1.available() > 0)
  { str = Serial1.readStringUntil('\n');// LF
    Serial.print ("Back: ");
    Serial.println (str);
  }
  digitalWrite(SetPin, HIGH);
  delay(100);
  Serial.println("Save & Exit");
}
#pragma once

#include <stdint.h>

class IrReceiver
{
  public:
    IrReceiver(uint8_t pin) :
      _pin(pin),
```

```
      _state(STATE_BEGIN),
      _data(0),
      _bit_counter(0)
    {
    }

    void init();

    void handleEdge();

    void handleTimeout();

    bool isDataReceived();

    uint16_t getReceivedData();

  private:
    enum State
    {
      STATE_BEGIN,
      STATE_START,
      STATE_DATA_STAGE_1,
      STATE_DATA_STAGE_2,
      STATE_CHECKING,
      STATE_RECEIVED
    };

  private:
    uint8_t _pin;
    volatile State _state;
    volatile uint16_t _data;
    volatile int _bit_counter;
};

#include "irreceiver.h"

#include <Arduino.h>

const unsigned int START_TIMEOUT = 2500;
const unsigned int DATA_STAGE_1_TIMEOUT = 1200;
const unsigned int DATA_STAGE_2_TIMEOUT = 900;
const unsigned int CHECKING_TIMEOUT = 10000;

static IrReceiver *active_receiver = NULL;

static void initTimer()
{
  TCCR4A = 0;
  TCCR4B = 0;
  TCCR4C = 0;
  TCCR4D = 0;
  TCCR4E = 0;
}

static void startTimer(unsigned int us)
{
  TC4H = 0;
  TCNT4 = 0;
```

```
  OCR4C = microsecondsToClockCycles(us) / 1024;
  TCCR4B = (1 << PSR4) |
           (1 << CS43) | (0 << CS42) |
           (1 << CS41) | (1 << CS40);
  TIMSK4 = (1 << TOIE4);
}

static void stopTimer()
{
  TIMSK4 = 0;
  TCCR4B = 0;
}

static void edgeHandler()
{
  if (active_receiver != NULL) {
    active_receiver->handleEdge();
  }
}

ISR(TIMER4_OVF_vect)
{
  if (active_receiver != NULL) {
    active_receiver->handleTimeout();
  }
}

void IrReceiver::init()
{
  _state = STATE_BEGIN;

  initTimer();

  active_receiver = this;

  pinMode(_pin, INPUT);

  attachInterrupt(digitalPinToInterrupt(_pin),
                  edgeHandler,
                  FALLING);
}

void IrReceiver::handleEdge()
{
  stopTimer();

  if (_state == STATE_BEGIN) {
    startTimer(START_TIMEOUT);

    _state = STATE_START;
    return;
  }

  if (_state == STATE_START) {
    _state = STATE_BEGIN;
    return;
  }
```

```
  if (_state == STATE_DATA_STAGE_1) {
    startTimer(DATA_STAGE_2_TIMEOUT);

    _state = STATE_DATA_STAGE_2;
    return;
  }

  if (_state == STATE_DATA_STAGE_2) {
    _state = STATE_BEGIN;
    return;
  }

  if (_state == STATE_CHECKING) {
    _state = STATE_BEGIN;
    return;
  }
}

void IrReceiver::handleTimeout()
{
  stopTimer();

  if (_state == STATE_START) {
    if (digitalRead(_pin) == HIGH) {
      startTimer(DATA_STAGE_1_TIMEOUT);
      _data = 0;
      _bit_counter = 12;

      _state = STATE_DATA_STAGE_1;
    } else {
      _state = STATE_BEGIN;
    }
    return;
  }

  if (_state == STATE_DATA_STAGE_1) {
    _state = STATE_BEGIN;
    return;
  }

  if (_state == STATE_DATA_STAGE_2) {
    if (digitalRead(_pin) == HIGH) {
      _data <<= 1;
    } else {
      _data <<= 1;
      _data |= 1;
    }

    _bit_counter--;

    if (_bit_counter > 0) {
      startTimer(DATA_STAGE_1_TIMEOUT);

      _state = STATE_DATA_STAGE_1;
    } else {
      startTimer(CHECKING_TIMEOUT);

      _state = STATE_CHECKING;
```

```
    }
    return;
  }

  if (_state == STATE_CHECKING) {
    _state = STATE_RECEIVED;
    return;
  }
}

bool IrReceiver::isDataReceived()
{
  cli();
  State state = _state;
  sei();
  return (state == STATE_RECEIVED);
}

uint16_t IrReceiver::getReceivedData()
{
  if (isDataReceived()) {
    cli();
    uint16_t data = _data;
    _state = STATE_BEGIN;
    sei();
    return data;
  } else {
    return 0;
  }
}
```

4.2 Подтрек «MariNet»

Задача:

В задаче по беспилотным плавательным устройствам вам нужно будет собрать и запрограммировать корабль с дифференциальным приводом, который должен преодолеть заданный водный маршрут, используя данные УЗ-датчиков, навигационные данные, и доставить груз в порт.

Базовый набор:

- Комплект для сборки «корабля» с системой крепления сервоприводов, сенсоров, груза, системы питания
- Управляющая плата Arduino
- Сервоприводы
- Радиомодуль
- Навигационная метка
- Комплект электронных компонентов
- Комплект крепежных элементов

**Полная и подробная комплектация предоставляется вместе с конструктором*

Электрическая схема:

Вариант принципиальной схемы подключения элементов бортового компьютера конструктора корабля:

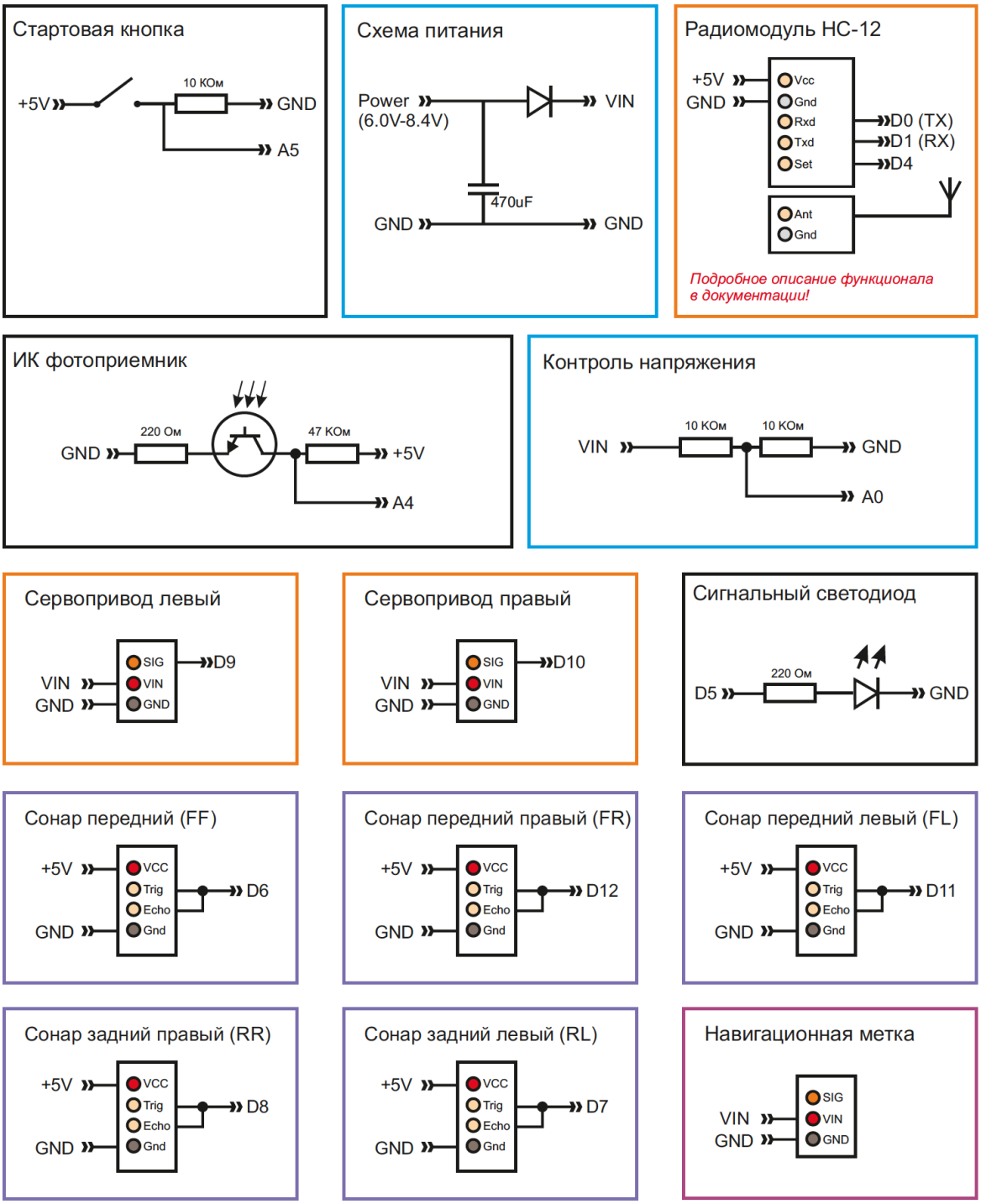
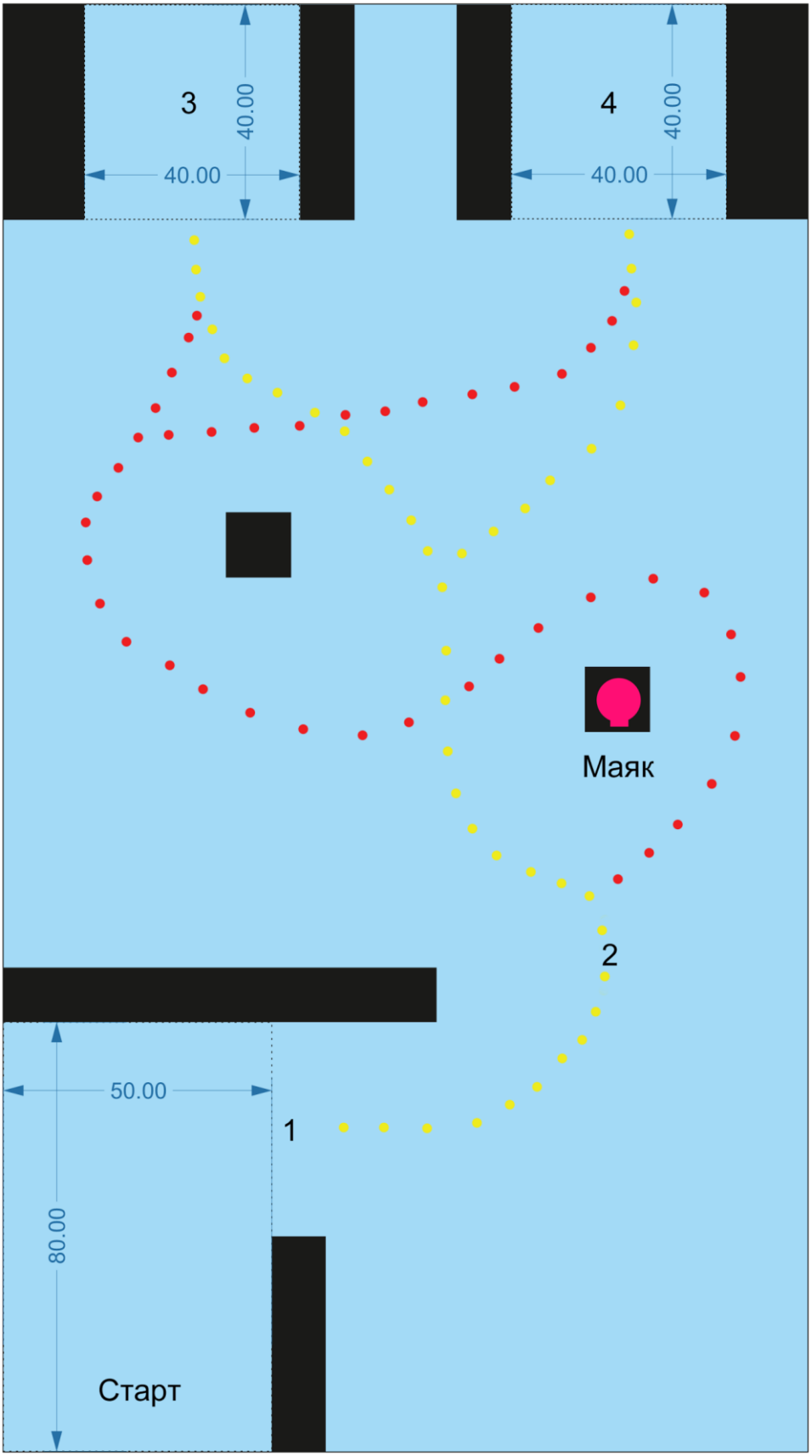


Схема полигона:

Полигон представляет собой бассейн размером 1.5м на 2.7м, наполненный водой и установленными блоками.

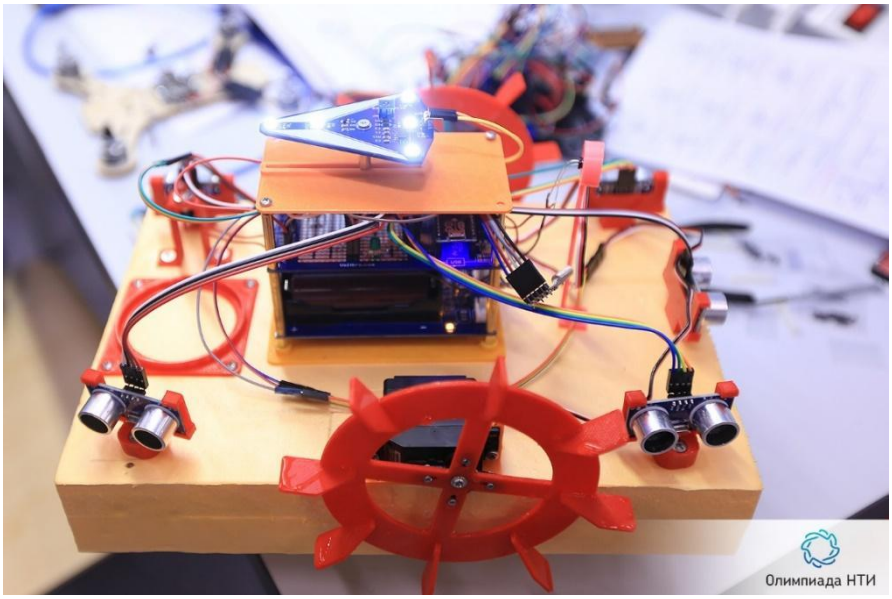

3
40.00
40.00
4
40.00
40.00
Маяк
2
50.00
1
80.00
Старт

Баллы и задачи:

	Задача	Баллы
1	Собрать модель Корабля: 1. Собрать корабль держащийся на воде без перекосов более 3 мм, не цепляющий лопастями за дно бассейна, несущий на себе всю электронику. Пройти отладочное тестирование узлов 2. Демонстрация работы лопастей 3. Демонстрация работы сонаров 4. Демонстрация работы ИК приемника 5. Демонстрация работы кнопки старта 6. Демонстрация работы контроля напряжения 7. Демонстрация работы радиомодуля 8. Демонстрация работы сигнального светодиода 9. Демонстрация работы навигационной метки Демонстрация работы датчиков осуществляется передачей отладочной информации на компьютер	0-9
2	Преодоление контрольной линии №1 После нажатия на кнопку старт, корабль должен начать выполнение задания. Задание считается выполненным если корабль пересекает линию №1. За касание бортов и блоков - штраф 1 балла За касание дна лопастями - штраф 1 балл Подзадачи: 1. Написать соответствующую заданию программу. 2. Предварительно выполнить задание 1 “Собрать корабль”	5
3	**Получение сигнала от маяка**, Необходимо встать на линию маяка и получить его сигнал (в этом задании маяк всегда включен), получение сигнала маяка демонстрируется включением сигнального светодиода и остановкой двигателей. Дополнительные Условия: 1. Касание борта - штраф 1 балл 3. Столкновение с маяком - штраф 3 балла	5
4	Доплыть до необходимого порта и остановиться. При включенном сигнале маяка плыть надо в порт №3. При выключенном маяке плыть надо в порт №4. Подзадачи: 1. Предварительно выполнить задания 1,2,3 2. С использованием спутника - бонус 4 балла 3. Заезд в порт со стороны кормы - бонус 3 балла 4. Приплыл не в тот порт - штраф 3 балла 5. Не выключил двигатели в порту - штраф 1 балл	17

	6. Касание борта или блока - штраф 1 балл 7. Столкновение с маяком - штраф 3 балла	
	Доплыть до необходимого порта по желтой траектории и остановиться. При включенном сигнале маяка плыть надо в порт №3. При выключенном маяке плыть надо в порт №4. 1. Приплыл не в тот порт - штраф 3 балла 2. Не выключил двигатели в порту - штраф 1 балл 3. Касание борта или блока - штраф 1 балл 4. Столкновение с маяком - штраф 3 балла	5
5.	Доплыть до необходимого порта по красной траектории и остановиться. При включенном сигнале маяка плыть надо в порт №3. При выключенном маяке плыть надо в порт №4. 5. Приплыл не в тот порт - штраф 3 балла 6. Не выключил двигатели в порту - штраф 1 балл 7. Касание борта или блока - штраф 1 балл 8. Столкновение с маяком - штраф 3 балла	9

Финальный вариант сборки устройства:



Полное решение:

```
#include <Servo.h>
#include <NewPing.h>

#define MAX_DISTANCE 200
```

```
#define HC12_SetupPin 4
#define GEO_CHANEL "AT+C001"
#define SAT_CHANEL "AT+C010"

Servo servoLeft, servoRight;

struct GEO {
  int x;
  int y;
  int grad;
};

GEO geo_data;


/*порты для оборудования*/
// Кнопка старта
int Button = 0;
const int ButtonPin = A5;

//передний дальнометр
const int FFtrig = 6;
const int FFecho = 6;
NewPing sonarFF(FFtrig,FFecho, MAX_DISTANCE);

//передний правый и задний правый дальнометры
const int FRtrig = 12;
const int FRecho = 12;
const int RRtrig = 8;
const int RRecho = 8;
NewPing sonarFR(FRtrig,FRecho, MAX_DISTANCE);
NewPing sonarRR(RRtrig,RRecho, MAX_DISTANCE);

//передний правый и задний правый дальнометры
const int FLtrig = 11;
const int FLecho = 11;
const int RLtrig = 7;
const int RLecho = 7;
NewPing sonarFL(FLtrig,FLecho, MAX_DISTANCE);
NewPing sonarRL(RLtrig,RLecho, MAX_DISTANCE);

float distFF;
float distFL;
float distFR;
float distRL;
float distRR;
float Light;
float Voltage;
float vLeft;
float vRight;


//светодатчики левый и правый
const int ir_pin = A4;
const int volt_pin = A0;

const int L_wheel = 9;
const int R_wheel = 10;
```

```
const int v = 10; //минимальна скорость
const int L = -1; //константа обозначающая поворот налево
const int R = 1; //константа обозначающая поворот направо
const int N = 0; //константа обознаяающая езду прямо
const int E = 2;//константа обозначающая достигание финиша, выключение
int k = N; //направление движения корабля, начальное значение
int I = 60; //расстояние от берега в см при котором нужно поворачивать

float turnSpeed = 70;
String str;




void setup()
{

  //сервопривод
  servo_attach();

  pinMode(ir_pin, INPUT);
  pinMode(volt_pin, INPUT);
  pinMode(ButtonPin, INPUT);

  Serial.begin(9600);
  Serial1.begin(9600);
  Serial1.flush();

  pinMode(HC12_SetupPin, OUTPUT);
  digitalWrite(HC12_SetupPin, HIGH);

  set_transmiter_chanel(GEO_CHANEL);


}


void loop()

{
  while (Button < 500) {
    Button = analogRead(ButtonPin);
    delay(10);
    k = N;
  }

  SensorScan();
  read_geo_data();

  SonarMove ();


  //CoordMove ();

}
```

```
float SonarMove ()
{

  vLeft = v;
  vRight = v;


  if(k == N)
  {
    if(distFF > 10)
    {
      if(distFL < 10) { vLeft += 0.2 * v; }
      if(distFR < 10) { vRight += 0.2 * v;}
      if(distFR > I && distRR > I  && distFL < I){ k = R;}
      if(distFL > I && distRL > I  && distFR < I){ k = L;}
    }

    if(distFF < 10)
    {
      if(distFL + distRL > distFR + distRR) {k = L;} else {k = R;}
    }


    if(distFL < 20 && distFF < 10 && distFR < 20 )
    {
      k = E;
    }

  }

  if(k == R)
  {
    servoLeft.write(0);
    servoRight.write(0);
    delay(1000);
    servoLeft.write(95);
    servoRight.write(85);
    delay(200);
    setServo(vLeft*10, vRight*10);
    delay(1500);
    k = N;
  }

  if(k == L)
  {
    servoLeft.write(180);
    servoRight.write(180);
    delay(1000);
    servoLeft.write(85);
    servoRight.write(95);
    delay(200);
    setServo(vLeft*10, vRight*10);
    delay(1500);
    k = N;
  }

  if(k == E)
  {
```

```
    vLeft = 0;
    vRight = 0;
    Button = 0;
  }

  setServo(vLeft,vRight);
  delay (50);
}

float CoordMove ()
{


  rotate_right_to(90);

  move_y(300);

  move_sat(25);

  move_y(300);

  rotate_right_to(180);

  move_x(-300);

  rotate_right_to(270);

  move_y(-300);

  move_sat(-25);

  move_y(-300);

  rotate_right_to(360);

  move_x(300);
}

float setServo(float vLeft, float vRight)
{
  if (vLeft > 90)  { vLeft = 90; }
  if (vRight > 90) { vRight = 90;}
  servoLeft.write(90 - vLeft);
  servoRight.write(90 + vRight);

}


float SensorScan ()
{

  distFF = sonarFF.ping_cm();
  delay(2);
  Serial.print("FF:");
  Serial.println(distFF);
  Serial.println("-");

  distFL = sonarFL.ping_cm();
```

```
  delay(2);
  Serial.print("FL:");
  Serial.println(distFL);

  distRL = sonarRL.ping_cm();
  delay(2);
  Serial.print("RL:");
  Serial.println(distRL);
  Serial.println("-");

  distFR = sonarFR.ping_cm();
  delay(2);
  Serial.print("FR:");
  Serial.println(distFR);

  distRR = sonarRR.ping_cm();
  delay(2);
  Serial.print("RR:");
  Serial.println(distRR);
  Serial.println("-");

  Light = analogRead(ir_pin);
  Serial.print("IR:");
  Serial.println(Light);

  Voltage = analogRead(volt_pin);
  Serial.print("Voltage:");
  Serial.println(Voltage);

}


void servo_detach() {

  servoLeft.detach();
  servoRight.detach();
}

void servo_attach() {

  servoLeft.attach(L_wheel);
  servoRight.attach(R_wheel);
  servo_stop();
}

void servo_stop() {

  servoLeft.write(90);
  servoRight.write(90);
}

GEO read_geo_data() {

  while (1) {

    if (Serial1.available() > 0) {

      str = Serial1.readStringUntil('\n');
```

```
      //str = "00;1019;0609;192";
      Serial.println(str);

      geo_data.x    = str.substring(3, 7).toInt();
      geo_data.y    = str.substring(8, 12).toInt();
      geo_data.grad = str.substring(13, 16).toInt();

      return geo_data;
    }
  }
}


void set_transmiter_chanel(char* set_chanel) {

  digitalWrite(HC12_SetupPin, LOW);
  delay(50);
  Send_command(set_chanel);
  digitalWrite(HC12_SetupPin, HIGH);

}

void Send_command(char* ATcommand) {

  Serial1.println (ATcommand);
  Serial.println (ATcommand);
  delay(200);
  if (Serial1.available() > 0)
  {
    str = Serial1.readStringUntil('\n');// LF
    Serial.println (str);
  }
  delay(100);
}



void move_sat(int steps) {

  //turn of servos
  servo_detach();

  set_transmiter_chanel(SAT_CHANEL);
  //STOP
  Serial1.write((byte) 0x00); delay(50); //first byte broken at radio
  Serial1.write((byte) 0x30); delay(50); // 0
  Serial1.println();
  delay(1000);

  Serial.println(String(steps));
  Serial1.println(String(steps));

  delay(1000);

  while (1) {
    if (Serial1.available() > 0) {

      str = Serial1.readStringUntil('\r'); //have /r/n sting
```

```
      Serial.println(str);
      if (str == "STOP") {
        set_transmiter_chanel(GEO_CHANEL);

        read_geo_data();//wait until geo data normal
        servo_attach();//attach servos

        return;
      }

    }
  }


}

void move_x(int steps) {

  GEO start_geo_data = read_geo_data();

  int speed = 30;

  servoRight.write(90 + speed);
  servoLeft.write(90 - speed);

  while (1) {

    geo_data = read_geo_data();

    if (steps > 0 and geo_data.x > start_geo_data.x + steps) {
      servo_stop();
      return;
    }

    if (steps < 0 and geo_data.x < start_geo_data.x + steps) {
      servo_stop();
      return;
    }
  }

}

void move_y(int steps) {

  GEO start_geo_data = read_geo_data();

  int speed = 30;

  servoRight.write(90 + speed);
  servoLeft.write(90 - speed);

  while (1) {

    geo_data = read_geo_data();

    if (steps > 0 and geo_data.y > start_geo_data.y + steps) {
      servo_stop();
      return;
```

```
    }

    if (steps < 0 and geo_data.y < start_geo_data.y + steps) {
      servo_stop();
      return;
    }
  }

}


void rotate_right_to(int grad) {

  int speed = 8;

  while (1) {
    geo_data = read_geo_data();

    whels_speed(-20, 20, 100);

    //aprox grad
    //TODO have problem near 360->0 position
    if (geo_data.grad >= grad - 4 and geo_data.grad <= grad + 4) {
      servo_stop();
      Serial.println("ROTATE END");
      return;
    }
  }

}

void whels_speed(int speed_l, int speed_r, int timeout) {

  servoLeft.write(90 + speed_l);
  servoRight.write(90 - speed_r);

  delay(timeout);
  servo_stop();

}
```

4.3 Подтрек «AeroNet»

Задача:

В задаче необходимо выполнить полетное задание, управляя квадрокоптером автономно, при помощи Arduino и УЗ-датчиков.

Полетное задание: осуществить автономной взлёт, автономное движение на определенном расстоянии от поверхности, обнаружение и облет препятствия, автономная посадка.

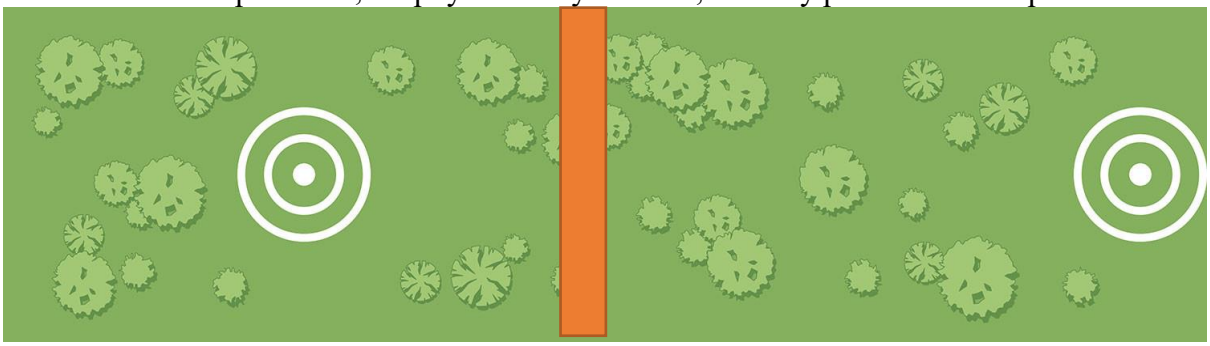
Базовый набор:

- Конструктор квадрокоптера Clever (рама, моторы, регуляторы, полетный контроллер, плата питания, пропеллеры, провода)
- Пульт радиоуправления
- Комплект электроники и датчиков

Схема полигона:

Стенд представляет собой ограниченное пространство (1.5м х 5.4 м).

Стены – оргстекло, сверху – натянута сетка, на полу расположено препятствие.

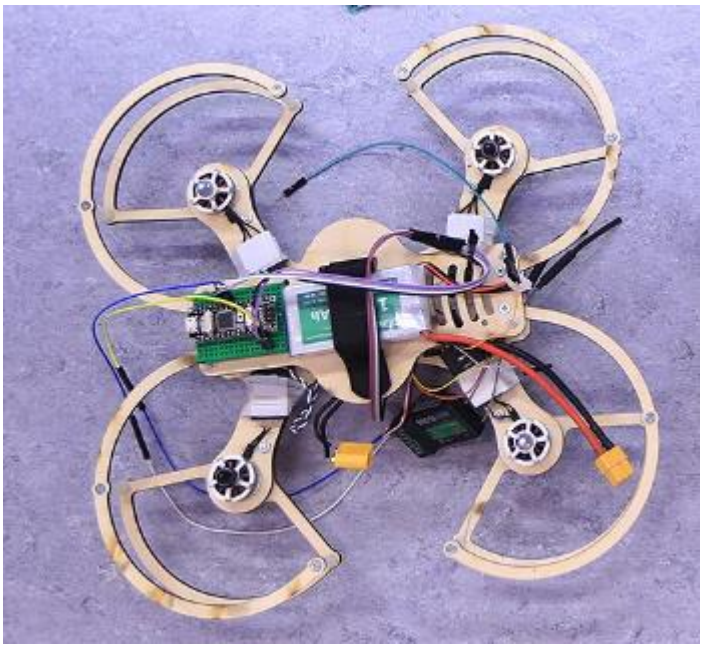


Баллы и задания:

	Задача	Баллы
1	Сборка коптера Собрать коптер, провести тестирование: ● Запуск по кнопке старта с пульта (2) ● Запуск моторов в автоматическом режиме (без винтов) (2) ● Работа сонаров (2) ● Крепление груза (2) ● Работа кнопки аварийного отключения	10
2	Под управление Arduino заставить коптер подняться в воздух на 50 см и приземлиться. Поломка защиты или шасси - штраф 3 балла, жесткая посадка - штраф 1 балл.	8
2	Стабилизировать коптер, используя ультразвуковые сонары. Подзадачи: ● запрограммировать коптер на взлёт, удержание высоты 50 см в течение: ○ 5 с -6 балла ○ 10 с – 8 балла ○ 20 с – 10 баллов ○ 30 с – 12 баллов ● запрограммировать коптер на взлёт, удержание высоты и позиции (расстояния от стен) и посадку	12
3	Перелететь через препятствие. Подзадачи: ● запрограммировать коптер на взлёт, преодоление препятствия и посадку ● не касаться стен и потолка во время выполнения полёта ● штраф за касание (2 балл, но не более 6	15

	штрафных баллов)	
4	Совершить посадку в обозначенную кругом зону ● красная зона (малый круг) -10 баллов ● зеленая зона (средний круг) – 7 балла ● синяя зона (большой круг) – 5 балла Подзадачи: не касаться стен и потолка во время выполнения полёта штраф за касание (1 балл, но не более 3х штрафных баллов)	10
5	Вернуться в зону старта. Подзадачи: ● запрограммировать коптер на взлёт, преодоление препятствия и посадку в зону «финиша» ● не касаться стен и потолка во время выполнения полёта ● совершить посадку в обозначенную кругом зону «старта»	5

Вариант финальной сборки устройства:



Полное решение:

```
#include "mspapi.h"

//-------------------------------------------------------------------
----------
```

```
//ЗАДАЧА
int wishY = 20;              // (см) желаемая высота
int wishX = 200;             // (см) желаемый путь
int tolerance = 3;           // (см) допустимая погрешность

// ПЕРЕМЕННЫЕ
// Подключение дальномеров
int pinTrig = 7;             // объединенный пин Trig на все
дальномеры
int pinEcho [] = {12, 6, 4}; // {pitchBack, pitchForward, throttle}
int wait = 250;              // время обработки показаний дальномера
int timeRefresh = 500;       // время обновления данных, передаваемых
на ПК

// Граничные значения моторов, отправляемых на ПК
//Тангаж
int pitchMin = 1000;
int pitchMax = 2000;
//int pitchCurrent = 1000;     // Текущее значение тангажа
int errorX = 0;              // (см) ошибка при движении по оси x
(путь)
int pitch = 1500;               // значение, которое передается на ПК
//Газ
int throttleMin = 1000;
int throttleMax = 2000;
//int throttleCurrent = 1000;  // Текущее значение тангажа
int errorY = 0;              // (см) ошибка при взлете/ посадке
int throttle = 0;            // значение, которое передается на ПК
// Крен
int roll = 1500;
//Рысканье
int yaw = 1500;


//БЛОКИРОВКА
bool armed = 1;
int pinRes = A1;             //чтобы активировать программу,
необходимо переключить потенциометр в крайнее положение
int pinLed = 10;             // оповещает о запуске программы


// ФУНКЦИЯ ОБРАБОТКИ СИГНАЛА ДАЛЬНОМЕРА
float distance (int Echo)
{
  // Запуск первой волны
  digitalWrite (pinTrig, LOW);
  delayMicroseconds (2);
  digitalWrite (pinTrig, HIGH);
  delayMicroseconds (10);
  digitalWrite (pinTrig, LOW);

  int t = pulseIn (Echo, HIGH); // (мкс) замер длины импульса (время
у/з за путь "туда-обратно")
  int space = t * 0.017;        // (см) вычисление расстояния до
препятствия
  if (space < 0) space = 0;     // если < 0 (бывает такой глюк), то 0
  // Serial.println (space);       // выводим расстояние на экран в
САНТИМЕТРАХ
```

```
  delay (wait);                  // задержка, чтобы волны у/з не
пересекались
  return (space);
}

// ФУНКЦИЯ ВЗЛЕТА/ПОСАДКИ
void takeOff (int kp, int wish)
{
  int currentHeight = distance (pinEcho[2]);
  Serial.println (currentHeight);
  errorY = currentHeight - wish;
  int up = -1 * (kp * errorY);
  throttle = throttle + up;
  // throttle = throttleCurrent;
  throttle = constrain (throttle, throttleMin, throttleMax);
  // Serial.println (throttle);
  return (throttle);
}

//ФУНКЦИЯ ОБРАТНОГО ОТСЧЕТА
void startLed (int count)
{
  digitalWrite (pinLed, HIGH);
  delay (2000);
  for (int i = 0; i < count; i++)
  {
    digitalWrite (pinLed, HIGH);
    delay (1000 - i * 100);
    digitalWrite (pinLed, LOW);
    delay (1000 - i * 100);
  }
}
//ФУНКЦИЯ РАЗБЛОКИРОВКИ
void unlockMotors ()
{ /*
    Чтобы разблокировать квадрик, необходимо выполнять следующую
последовательность действий
    1) Арм 0, газ - минимальный
    2) Арм 1, газ - минимальный
    3) Арм 1, газ - средний
  */

  // Цикл нужен для отправки пакета импульсов
  for (int j = 0; j < 20; j++)
  {
    armed = 0;
    throttle = throttleMin;
    pitch = pitchMin;
    msp::send_rc_command(yaw, pitch, roll, throttle, armed);
    delay (10);
  }
  delay (1000);

  for (int j = 0; j < 20; j++)
  {
    armed = 1;
    throttle = throttleMin;
    pitch = pitchMin;
```

```
    msp::send_rc_command(yaw, pitch, roll, throttle, armed);
   delay (10);
  }
  delay (1000);

  for (int j = 0; j < 20; j++)
  {
    armed = 1;
    throttle = throttleMin + 500;
    pitch = pitchMin;
    msp::send_rc_command(yaw, pitch, roll, throttle, armed);
    delay (10);
  }
  delay (1000);
}


// ======================     ОСНОВНАЯ  ПРОГРАММА
============================
void setup() {
  msp::setup();
  for (int i = 0; i < 3; i++)
    pinMode (pinEcho[i], INPUT);
  pinMode (pinTrig, OUTPUT);
  pinMode (pinLed, OUTPUT);
  pinMode (pinRes, INPUT);
  Serial.begin (115200);                // в оригинале скорость
передачи стоит (115200)

  unlockMotors ();
}

void loop() {
  // Получаем данные с ДАЛЬНОМЕРОВ
  // int d1 = distance (pinEcho[0]);
  // int d2 = distance (pinEcho[1]);
  // int d3 = distance (pinEcho[2]);
  Serial.println ();
  int value = analogRead (pinRes);
  Serial.println (value);
  if (value > 800)
  {
    startLed (5);                       // светодиод моргнет 5 раз,
затем начнет выполнятся программа


    // ВЗЛЕТ
    while (distance (pinEcho[2]) < wishY - tolerance or distance
(pinEcho[2]) > wishY + tolerance )
    {
      int i = 10;
      takeOff (10, wishY);
      Serial.println (throttle);
      while (i > 0)
      {
        msp::send_rc_command(yaw, pitch, roll, throttle, armed);
// ОТПРАВКА на   ПК
        i = i - 1;
```

```
        delay (10);
      }
      delay (timeRefresh);
    }
    Serial.println ("DONE!!!!");
    delay (5000);                    // время зависания
    Serial.println ("Landing...");

    //ПОСАДКА
    while (distance (pinEcho[2]) > 2 + tolerance )
    {
      int i = 10;
      takeOff (10, 5);
      Serial.println (throttle);
      while (i > 0)
      {
        msp::send_rc_command(yaw, pitch, roll, throttle, armed);
// ОТПРАВКА на   ПК
        i = i - 1;
        delay (10);
      }
      delay (timeRefresh);
    }
    Serial.println ("LANDING DONE ");
    delay (15000);
  }
}
#include "mspapi.h"

namespace msp
{

namespace attitude
{

int16_t pitch = 0;
int16_t roll = 0;
int16_t yaw = 0;

} // namespace sensors

namespace state
{

MSP_Parser parser;

} // namespace state

namespace handler
{

proto::rc rc;
proto::attitude attitude;

} // namespace handler

} // namespace msp
#ifndef _HEADER_7R3NEIGWH492XNOQPOQW_INCLUDED_
```

```
#define _HEADER_7R3NEIGWH492XNOQPOQW_INCLUDED_

#include "msppg.h"
#include <inttypes.h>
#if defined(ARDUINO) && ARDUINO >= 100
  #include "Arduino.h"
#else
  #include "WProgram.h"
#endif

/* Канал контроллера, отвечающий за предохранитель двигателей */
#define MSPPG_ARM_AUX 1

namespace msp
{

/* Инициализация парсера */
inline void setup();

/* Отправка управляющей команды на полетный контроллер */
inline void send_rc_command(int yaw, int pitch, int roll, int
throttle,
                            bool armed = true);

/* Отправка на порт произвольной строки из 15 символов, которая будет
 * принята полетным контроллером как пакет и будет проигнорована */
inline void send_debug(const char s[15]);

/* Отправка запроса на получение текущих установок контроллера */
inline void send_rc_request();

/* Отправка запроса на получение значений акселерометра
 * После получения, результаты будут отправлены в sensors::accel_x,
 * sensors::accel_y, sensors::accel_heading */
inline void send_attitude_request();

/* Чтение и обработка пакетов, пришедших на порт с полетного
контроллера */
inline void read_replies();


namespace attitude
{

extern int16_t pitch;
extern int16_t roll;
extern int16_t yaw;

} // namespace sensors

namespace state
{

extern MSP_Parser parser;

} // namespace state

namespace handler
```

```
{
namespace proto
{
class rc : public RC_Handler
{
public:
  void handle_RC(short c1, short c2, short c3, short c4, short c5,
short c6,
                 short c7, short c8, short c9, short c10, short c11,
short c12,
                 short c13, short c14, short c15, short c16, short
c17, short c18)
  {
    msp::send_debug("rc recv");
  }
};

class attitude : public ATTITUDE_Handler
{
public:
  void handle_ATTITUDE(short angx, short angy, short heading) override
  {
    msp::attitude::pitch = angx;
    msp::attitude::roll = angy;
    msp::attitude::yaw = heading;
    msp::send_debug("attitude");
  }
};
} // namespace proto

extern proto::rc rc;
extern proto::attitude attitude;

} // namespace handler
} // namespace msp

void msp::send_rc_command(int yaw, int pitch, int roll, int throttle,
bool armed)
{
#if MSPPG_ARM_AUX == 1
  MSP_Message msg = state::parser.serialize_SET_RAW_RC(roll, pitch,
throttle,
  yaw, armed ? 1800 : 1000, 1000, 1000, 1000);
#elif MSPPG_ARM_AUX == 2
  MSP_Message msg = state::parser.serialize_SET_RAW_RC(roll, pitch,
throttle,
  yaw, 1000, armed ? 1800 : 1000, 1000, 1000);
#else
  #error Unsupported AUX channel! Check MSPPG_ARM_AUX value.
#endif

  Serial.write(msg.data(), msg.length());
}

void msp::send_attitude_request()
{
  MSP_Message msg = state::parser.serialize_ATTITUDE_Request();
  Serial.write(msg.data(), msg.length());
```

```
}

void msp::send_rc_request()
{
  MSP_Message msg = state::parser.serialize_RC_Request();
  Serial.write(msg.data(), msg.length());
}

void msp::send_debug(const char s[15])
{
  MSP_Message msg = state::parser.serialize_DEBUG(s[0], s[1], s[2],
s[3], s[4],
  s[5], s[6], s[7], s[8], s[9], s[10], s[11], s[12], s[13], s[14],
'\n');

  Serial.write(msg.data(), msg.length());
}

void msp::setup()
{
  // cleanflight parser config
  state::parser.set_RC_Handler(&handler::rc);
  state::parser.set_ATTITUDE_Handler(&handler::attitude);
}

void msp::read_replies()
{
  // read all data from serial
  while (Serial.available() > 0)
  {
    state::parser.parse(Serial.read());
  }
}

#endif // _HEADER_7R3NEIGWH492XNOQPOQW_INCLUDED_
// AUTO-GENERATED CODE: DO NOT EDIT!!!


#include "msppg.h"

#include <string.h>
#include <stdio.h>
#include <stdlib.h>

static byte CRC8(byte * data, int n) {

    byte crc = 0x00;

    for (int k=0; k<n; ++k) {

        crc ^= data[k];
    }

    return crc;
}

byte MSP_Message::start() {
```

```
    this->pos = 0;
    return this->getNext();
}

bool MSP_Message::hasNext() {

    return this->pos <= this->len;
}


byte MSP_Message::getNext() {

    return this->bytes[this->pos++];
}

MSP_Parser::MSP_Parser() {

    this->state = 0;
}

void MSP_Parser::parse(byte b) {

    switch (this->state) {

        case 0:                 // sync char 1
            if (b == 36) { // $
                this->state++;
            }
            break;

        case 1:                 // sync char 2
            if (b == 77) { // M
                this->state++;
            }
            else {              // restart and try again
                this->state = 0;
            }
            break;

        case 2:                 // direction (should be >)
            if (b == 62) { // >
                this->message_direction = 1;
            }
            else {              // <
                this->message_direction = 0;
            }
            this->state++;
            break;

        case 3:
            this->message_length_expected = b;
            this->message_checksum = b;
            // setup arraybuffer
            this->message_length_received = 0;
            this->state++;
            break;

        case 4:
```

```
            this->message_id = b;
            this->message_checksum ^= b;
            if (this->message_length_expected > 0) {
                // process payload
                this->state++;
            }
            else {
                // no payload
                this->state += 2;
            }
            break;

        case 5: // payload
            this->message_buffer[this->message_length_received] = b;
            this->message_checksum ^= b;
            this->message_length_received++;
            if (this->message_length_received >= this-
>message_length_expected) {
                this->state++;
            }
            break;

        case 6:
            this->state = 0;
            if (this->message_checksum == b) {
                // message received, process
                switch (this->message_id) {

                    case 105: {

                        short c1;
                        memcpy(&c1,  &this->message_buffer[0],
sizeof(short));

                        short c2;
                        memcpy(&c2,  &this->message_buffer[2],
sizeof(short));

                        short c3;
                        memcpy(&c3,  &this->message_buffer[4],
sizeof(short));

                        short c4;
                        memcpy(&c4,  &this->message_buffer[6],
sizeof(short));

                        short c5;
                        memcpy(&c5,  &this->message_buffer[8],
sizeof(short));

                        short c6;
                        memcpy(&c6,  &this->message_buffer[10],
sizeof(short));

                        short c7;
                        memcpy(&c7,  &this->message_buffer[12],
sizeof(short));
```

```
                        short c8;
                        memcpy(&c8,  &this->message_buffer[14],
sizeof(short));

                        short c9;
                        memcpy(&c9,  &this->message_buffer[16],
sizeof(short));

                        short c10;
                        memcpy(&c10,  &this->message_buffer[18],
sizeof(short));

                        short c11;
                        memcpy(&c11,  &this->message_buffer[20],
sizeof(short));

                        short c12;
                        memcpy(&c12,  &this->message_buffer[22],
sizeof(short));

                        short c13;
                        memcpy(&c13,  &this->message_buffer[24],
sizeof(short));

                        short c14;
                        memcpy(&c14,  &this->message_buffer[26],
sizeof(short));

                        short c15;
                        memcpy(&c15,  &this->message_buffer[28],
sizeof(short));

                        short c16;
                        memcpy(&c16,  &this->message_buffer[30],
sizeof(short));

                        short c17;
                        memcpy(&c17,  &this->message_buffer[32],
sizeof(short));

                        short c18;
                        memcpy(&c18,  &this->message_buffer[34],
sizeof(short));

                        this->handlerForRC->handle_RC(c1, c2, c3, c4,
c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16, c17, c18);
                        } break;

                    case 108: {

                        short angx;
                        memcpy(&angx,  &this->message_buffer[0],
sizeof(short));

                        short angy;
                        memcpy(&angy,  &this->message_buffer[2],
sizeof(short));
```

```
                        short heading;
                        memcpy(&heading,  &this->message_buffer[4],
sizeof(short));

                        this->handlerForATTITUDE-
>handle_ATTITUDE(angx, angy, heading);
                        } break;

                    case 104: {

                        short c1;
                        memcpy(&c1,  &this->message_buffer[0],
sizeof(short));

                        short c2;
                        memcpy(&c2,  &this->message_buffer[2],
sizeof(short));

                        short c3;
                        memcpy(&c3,  &this->message_buffer[4],
sizeof(short));

                        short c4;
                        memcpy(&c4,  &this->message_buffer[6],
sizeof(short));

                        short c5;
                        memcpy(&c5,  &this->message_buffer[8],
sizeof(short));

                        short c6;
                        memcpy(&c6,  &this->message_buffer[10],
sizeof(short));

                        short c7;
                        memcpy(&c7,  &this->message_buffer[12],
sizeof(short));

                        short c8;
                        memcpy(&c8,  &this->message_buffer[14],
sizeof(short));

                        this->handlerForMOTOR->handle_MOTOR(c1, c2,
c3, c4, c5, c6, c7, c8);
                        } break;

                    case 125: {

                        short framecount;
                        memcpy(&framecount,  &this->message_buffer[0],
sizeof(short));

                        short pixel_flow_x_sum;
                        memcpy(&pixel_flow_x_sum,  &this-
>message_buffer[2], sizeof(short));

                        short pixel_flow_y_sum;
                        memcpy(&pixel_flow_y_sum,  &this-
```

```
>message_buffer[4], sizeof(short));

                        short flow_comp_m_x;
                        memcpy(&flow_comp_m_x,  &this-
>message_buffer[6], sizeof(short));

                        short flow_comp_m_y;
                        memcpy(&flow_comp_m_y,  &this-
>message_buffer[8], sizeof(short));

                        short qual;
                        memcpy(&qual,  &this->message_buffer[10],
sizeof(short));

                        short gyro_x_rate;
                        memcpy(&gyro_x_rate,  &this-
>message_buffer[12], sizeof(short));

                        short gyro_y_rate;
                        memcpy(&gyro_y_rate,  &this-
>message_buffer[14], sizeof(short));

                        short gyro_z_rate;
                        memcpy(&gyro_z_rate,  &this-
>message_buffer[16], sizeof(short));

                        byte gyro_range;
                        memcpy(&gyro_range,  &this-
>message_buffer[18], sizeof(byte));

                        byte sonar_timestamp;
                        memcpy(&sonar_timestamp,  &this-
>message_buffer[19], sizeof(byte));

                        short ground_distance;
                        memcpy(&ground_distance,  &this-
>message_buffer[20], sizeof(short));

                        this->handlerForPX4FLOW-
>handle_PX4FLOW(framecount, pixel_flow_x_sum, pixel_flow_y_sum,
flow_comp_m_x, flow_comp_m_y, qual, gyro_x_rate, gyro_y_rate,
gyro_z_rate, gyro_range, sonar_timestamp, ground_distance);
                        } break;

                    case 101: {

                        short cycleTime;
                        memcpy(&cycleTime,  &this->message_buffer[0],
sizeof(short));

                        short i2c_errors_count;
                        memcpy(&i2c_errors_count,  &this-
>message_buffer[2], sizeof(short));

                        short sensor;
                        memcpy(&sensor,  &this->message_buffer[4],
sizeof(short));
```

```
                        int flag;
                        memcpy(&flag,  &this->message_buffer[6],
sizeof(int));

                        byte currentSet;
                        memcpy(&currentSet,  &this-
>message_buffer[10], sizeof(byte));

                        this->handlerForMSP_STATUS-
>handle_MSP_STATUS(cycleTime, i2c_errors_count, sensor, flag,
currentSet);
                        } break;

                    case 188: {

                        byte c1;
                        memcpy(&c1,  &this->message_buffer[0],
sizeof(byte));

                        byte c2;
                        memcpy(&c2,  &this->message_buffer[1],
sizeof(byte));

                        byte c3;
                        memcpy(&c3,  &this->message_buffer[2],
sizeof(byte));

                        byte c4;
                        memcpy(&c4,  &this->message_buffer[3],
sizeof(byte));

                        byte c5;
                        memcpy(&c5,  &this->message_buffer[4],
sizeof(byte));

                        byte c6;
                        memcpy(&c6,  &this->message_buffer[5],
sizeof(byte));

                        byte c7;
                        memcpy(&c7,  &this->message_buffer[6],
sizeof(byte));

                        byte c8;
                        memcpy(&c8,  &this->message_buffer[7],
sizeof(byte));

                        byte c9;
                        memcpy(&c9,  &this->message_buffer[8],
sizeof(byte));

                        byte c10;
                        memcpy(&c10,  &this->message_buffer[9],
sizeof(byte));

                        byte c11;
                        memcpy(&c11,  &this->message_buffer[10],
sizeof(byte));
```

```
                        byte c12;
                        memcpy(&c12,  &this->message_buffer[11],
sizeof(byte));

                        byte c13;
                        memcpy(&c13,  &this->message_buffer[12],
sizeof(byte));

                        byte c14;
                        memcpy(&c14,  &this->message_buffer[13],
sizeof(byte));

                        byte c15;
                        memcpy(&c15,  &this->message_buffer[14],
sizeof(byte));

                        byte c16;
                        memcpy(&c16,  &this->message_buffer[15],
sizeof(byte));

                        this->handlerForDEBUG->handle_DEBUG(c1, c2,
c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, c15, c16);
                        } break;

                }
            }

            break;

        default:
            break;
    }
}

void MSP_Parser::set_RC_Handler(class RC_Handler * handler) {

    this->handlerForRC = handler;
}

MSP_Message MSP_Parser::serialize_RC_Request() {

    MSP_Message msg;

    msg.bytes[0] = 36;
    msg.bytes[1] = 77;
    msg.bytes[2] = 60;
    msg.bytes[3] = 0;
    msg.bytes[4] = 105;
    msg.bytes[5] = 105;

    msg.len = 6;

    return msg;
}MSP_Message MSP_Parser::serialize_RC(short c1, short c2, short c3,
short c4, short c5, short c6, short c7, short c8, short c9, short c10,
short c11, short c12, short c13, short c14, short c15, short c16,
short c17, short c18) {
```

```
    MSP_Message msg;

    msg.bytes[0] = 36;
    msg.bytes[1] = 77;
    msg.bytes[2] = 60;
    msg.bytes[3] = 36;
    msg.bytes[4] = 105;

    memcpy(&msg.bytes[5], &c1, sizeof(short));
    memcpy(&msg.bytes[7], &c2, sizeof(short));
    memcpy(&msg.bytes[9], &c3, sizeof(short));
    memcpy(&msg.bytes[11], &c4, sizeof(short));
    memcpy(&msg.bytes[13], &c5, sizeof(short));
    memcpy(&msg.bytes[15], &c6, sizeof(short));
    memcpy(&msg.bytes[17], &c7, sizeof(short));
    memcpy(&msg.bytes[19], &c8, sizeof(short));
    memcpy(&msg.bytes[21], &c9, sizeof(short));
    memcpy(&msg.bytes[23], &c10, sizeof(short));
    memcpy(&msg.bytes[25], &c11, sizeof(short));
    memcpy(&msg.bytes[27], &c12, sizeof(short));
    memcpy(&msg.bytes[29], &c13, sizeof(short));
    memcpy(&msg.bytes[31], &c14, sizeof(short));
    memcpy(&msg.bytes[33], &c15, sizeof(short));
    memcpy(&msg.bytes[35], &c16, sizeof(short));
    memcpy(&msg.bytes[37], &c17, sizeof(short));
    memcpy(&msg.bytes[39], &c18, sizeof(short));

    msg.bytes[41] = CRC8(&msg.bytes[3], 38);

    msg.len = 42;

    return msg;
}

void MSP_Parser::set_ATTITUDE_Handler(class ATTITUDE_Handler *
handler) {

    this->handlerForATTITUDE = handler;
}

MSP_Message MSP_Parser::serialize_ATTITUDE_Request() {

    MSP_Message msg;

    msg.bytes[0] = 36;
    msg.bytes[1] = 77;
    msg.bytes[2] = 60;
    msg.bytes[3] = 0;
    msg.bytes[4] = 108;
    msg.bytes[5] = 108;

    msg.len = 6;

    return msg;
}MSP_Message MSP_Parser::serialize_ATTITUDE(short angx, short angy,
short heading) {
```

```
    MSP_Message msg;

    msg.bytes[0] = 36;
    msg.bytes[1] = 77;
    msg.bytes[2] = 60;
    msg.bytes[3] = 6;
    msg.bytes[4] = 108;

    memcpy(&msg.bytes[5], &angx, sizeof(short));
    memcpy(&msg.bytes[7], &angy, sizeof(short));
    memcpy(&msg.bytes[9], &heading, sizeof(short));

    msg.bytes[11] = CRC8(&msg.bytes[3], 8);

    msg.len = 12;

    return msg;
}

MSP_Message MSP_Parser::serialize_SET_MOTOR(short c1, short c2, short
c3, short c4, short c5, short c6, short c7, short c8) {

    MSP_Message msg;

    msg.bytes[0] = 36;
    msg.bytes[1] = 77;
    msg.bytes[2] = 60;
    msg.bytes[3] = 16;
    msg.bytes[4] = 214;

    memcpy(&msg.bytes[5], &c1, sizeof(short));
    memcpy(&msg.bytes[7], &c2, sizeof(short));
    memcpy(&msg.bytes[9], &c3, sizeof(short));
    memcpy(&msg.bytes[11], &c4, sizeof(short));
    memcpy(&msg.bytes[13], &c5, sizeof(short));
    memcpy(&msg.bytes[15], &c6, sizeof(short));
    memcpy(&msg.bytes[17], &c7, sizeof(short));
    memcpy(&msg.bytes[19], &c8, sizeof(short));

    msg.bytes[21] = CRC8(&msg.bytes[3], 18);

    msg.len = 22;

    return msg;
}

void MSP_Parser::set_MOTOR_Handler(class MOTOR_Handler * handler) {

    this->handlerForMOTOR = handler;
}

MSP_Message MSP_Parser::serialize_MOTOR_Request() {

    MSP_Message msg;

    msg.bytes[0] = 36;
    msg.bytes[1] = 77;
    msg.bytes[2] = 60;
```

```
    msg.bytes[3] = 0;
    msg.bytes[4] = 104;
    msg.bytes[5] = 104;

    msg.len = 6;

    return msg;
}MSP_Message MSP_Parser::serialize_MOTOR(short c1, short c2, short c3,
short c4, short c5, short c6, short c7, short c8) {

    MSP_Message msg;

    msg.bytes[0] = 36;
    msg.bytes[1] = 77;
    msg.bytes[2] = 60;
    msg.bytes[3] = 16;
    msg.bytes[4] = 104;

    memcpy(&msg.bytes[5], &c1, sizeof(short));
    memcpy(&msg.bytes[7], &c2, sizeof(short));
    memcpy(&msg.bytes[9], &c3, sizeof(short));
    memcpy(&msg.bytes[11], &c4, sizeof(short));
    memcpy(&msg.bytes[13], &c5, sizeof(short));
    memcpy(&msg.bytes[15], &c6, sizeof(short));
    memcpy(&msg.bytes[17], &c7, sizeof(short));
    memcpy(&msg.bytes[19], &c8, sizeof(short));

    msg.bytes[21] = CRC8(&msg.bytes[3], 18);

    msg.len = 22;

    return msg;
}

void MSP_Parser::set_PX4FLOW_Handler(class PX4FLOW_Handler * handler)
{

    this->handlerForPX4FLOW = handler;
}

MSP_Message MSP_Parser::serialize_PX4FLOW_Request() {

    MSP_Message msg;

    msg.bytes[0] = 36;
    msg.bytes[1] = 77;
    msg.bytes[2] = 60;
    msg.bytes[3] = 0;
    msg.bytes[4] = 125;
    msg.bytes[5] = 125;

    msg.len = 6;

    return msg;
}MSP_Message MSP_Parser::serialize_PX4FLOW(short framecount, short
pixel_flow_x_sum, short pixel_flow_y_sum, short flow_comp_m_x, short
flow_comp_m_y, short qual, short gyro_x_rate, short gyro_y_rate, short
gyro_z_rate, byte gyro_range, byte sonar_timestamp, short
```

```
ground_distance) {

    MSP_Message msg;

    msg.bytes[0] = 36;
    msg.bytes[1] = 77;
    msg.bytes[2] = 60;
    msg.bytes[3] = 22;
    msg.bytes[4] = 125;

    memcpy(&msg.bytes[5], &framecount, sizeof(short));
    memcpy(&msg.bytes[7], &pixel_flow_x_sum, sizeof(short));
    memcpy(&msg.bytes[9], &pixel_flow_y_sum, sizeof(short));
    memcpy(&msg.bytes[11], &flow_comp_m_x, sizeof(short));
    memcpy(&msg.bytes[13], &flow_comp_m_y, sizeof(short));
    memcpy(&msg.bytes[15], &qual, sizeof(short));
    memcpy(&msg.bytes[17], &gyro_x_rate, sizeof(short));
    memcpy(&msg.bytes[19], &gyro_y_rate, sizeof(short));
    memcpy(&msg.bytes[21], &gyro_z_rate, sizeof(short));
    memcpy(&msg.bytes[23], &gyro_range, sizeof(byte));
    memcpy(&msg.bytes[24], &sonar_timestamp, sizeof(byte));
    memcpy(&msg.bytes[25], &ground_distance, sizeof(short));

    msg.bytes[27] = CRC8(&msg.bytes[3], 24);

    msg.len = 28;

    return msg;
}

MSP_Message MSP_Parser::serialize_SET_RAW_RC(short c1, short c2, short
c3, short c4, short c5, short c6, short c7, short c8) {

    MSP_Message msg;

    msg.bytes[0] = 36;
    msg.bytes[1] = 77;
    msg.bytes[2] = 60;
    msg.bytes[3] = 16;
    msg.bytes[4] = 200;

    memcpy(&msg.bytes[5], &c1, sizeof(short));
    memcpy(&msg.bytes[7], &c2, sizeof(short));
    memcpy(&msg.bytes[9], &c3, sizeof(short));
    memcpy(&msg.bytes[11], &c4, sizeof(short));
    memcpy(&msg.bytes[13], &c5, sizeof(short));
    memcpy(&msg.bytes[15], &c6, sizeof(short));
    memcpy(&msg.bytes[17], &c7, sizeof(short));
    memcpy(&msg.bytes[19], &c8, sizeof(short));

    msg.bytes[21] = CRC8(&msg.bytes[3], 18);

    msg.len = 22;

    return msg;
}

void MSP_Parser::set_MSP_STATUS_Handler(class MSP_STATUS_Handler *
```

```
handler) {

    this->handlerForMSP_STATUS = handler;
}

MSP_Message MSP_Parser::serialize_MSP_STATUS_Request() {

    MSP_Message msg;

    msg.bytes[0] = 36;
    msg.bytes[1] = 77;
    msg.bytes[2] = 60;
    msg.bytes[3] = 0;
    msg.bytes[4] = 101;
    msg.bytes[5] = 101;

    msg.len = 6;

    return msg;
}MSP_Message MSP_Parser::serialize_MSP_STATUS(short cycleTime, short
i2c_errors_count, short sensor, int flag, byte currentSet) {

    MSP_Message msg;

    msg.bytes[0] = 36;
    msg.bytes[1] = 77;
    msg.bytes[2] = 60;
    msg.bytes[3] = 11;
    msg.bytes[4] = 101;

    memcpy(&msg.bytes[5], &cycleTime, sizeof(short));
    memcpy(&msg.bytes[7], &i2c_errors_count, sizeof(short));
    memcpy(&msg.bytes[9], &sensor, sizeof(short));
    memcpy(&msg.bytes[11], &flag, sizeof(int));
    memcpy(&msg.bytes[15], &currentSet, sizeof(byte));

    msg.bytes[16] = CRC8(&msg.bytes[3], 13);

    msg.len = 17;

    return msg;
}

void MSP_Parser::set_DEBUG_Handler(class DEBUG_Handler * handler) {

    this->handlerForDEBUG = handler;
}

MSP_Message MSP_Parser::serialize_DEBUG_Request() {

    MSP_Message msg;

    msg.bytes[0] = 36;
    msg.bytes[1] = 77;
    msg.bytes[2] = 60;
    msg.bytes[3] = 0;
    msg.bytes[4] = 188;
    msg.bytes[5] = 188;
```

```
    msg.len = 6;

    return msg;
}MSP_Message MSP_Parser::serialize_DEBUG(byte c1, byte c2, byte c3,
byte c4, byte c5, byte c6, byte c7, byte c8, byte c9, byte c10, byte
c11, byte c12, byte c13, byte c14, byte c15, byte c16) {

    MSP_Message msg;

    msg.bytes[0] = 36;
    msg.bytes[1] = 77;
    msg.bytes[2] = 60;
    msg.bytes[3] = 16;
    msg.bytes[4] = 188;

    memcpy(&msg.bytes[5], &c1, sizeof(byte));
    memcpy(&msg.bytes[6], &c2, sizeof(byte));
    memcpy(&msg.bytes[7], &c3, sizeof(byte));
    memcpy(&msg.bytes[8], &c4, sizeof(byte));
    memcpy(&msg.bytes[9], &c5, sizeof(byte));
    memcpy(&msg.bytes[10], &c6, sizeof(byte));
    memcpy(&msg.bytes[11], &c7, sizeof(byte));
    memcpy(&msg.bytes[12], &c8, sizeof(byte));
    memcpy(&msg.bytes[13], &c9, sizeof(byte));
    memcpy(&msg.bytes[14], &c10, sizeof(byte));
    memcpy(&msg.bytes[15], &c11, sizeof(byte));
    memcpy(&msg.bytes[16], &c12, sizeof(byte));
    memcpy(&msg.bytes[17], &c13, sizeof(byte));
    memcpy(&msg.bytes[18], &c14, sizeof(byte));
    memcpy(&msg.bytes[19], &c15, sizeof(byte));
    memcpy(&msg.bytes[20], &c16, sizeof(byte));

    msg.bytes[21] = CRC8(&msg.bytes[3], 18);

    msg.len = 22;

    return msg;
}
// AUTO-GENERATED CODE: DO NOT EDIT!!!\n\n'

static const int MAXBUF = 64;

typedef unsigned char byte;

class MSP_Message {

    friend class MSP_Parser;

    protected:

        MSP_Message() { }
        byte bytes[MAXBUF];
        int pos;
        int len;

    public:
        inline int length() const { return len; }
```

```
        inline byte* data() { return bytes; }

        byte start();
        bool hasNext();
        byte getNext();

};

class MSP_Parser {

    private:

        int state;
        byte message_direction;
        byte message_id;
        byte message_length_expected;
        byte message_length_received;
        byte message_buffer[MAXBUF];
        byte message_checksum;

    public:

        MSP_Parser();

        void parse(byte b);


        MSP_Message serialize_RC(short c1, short c2, short c3, short
c4, short c5, short c6, short c7, short c8, short c9, short c10, short
c11, short c12, short c13, short c14, short c15, short c16, short c17,
short c18);

        MSP_Message serialize_RC_Request();

        void set_RC_Handler(class RC_Handler * handler);

        MSP_Message serialize_ATTITUDE(short angx, short angy, short
heading);

        MSP_Message serialize_ATTITUDE_Request();

        void set_ATTITUDE_Handler(class ATTITUDE_Handler * handler);

        MSP_Message serialize_SET_MOTOR(short c1, short c2, short c3,
short c4, short c5, short c6, short c7, short c8);

        MSP_Message serialize_MOTOR(short c1, short c2, short c3,
short c4, short c5, short c6, short c7, short c8);

        MSP_Message serialize_MOTOR_Request();

        void set_MOTOR_Handler(class MOTOR_Handler * handler);

        MSP_Message serialize_PX4FLOW(short framecount, short
pixel_flow_x_sum, short pixel_flow_y_sum, short flow_comp_m_x, short
flow_comp_m_y, short qual, short gyro_x_rate, short gyro_y_rate, short
gyro_z_rate, byte gyro_range, byte sonar_timestamp, short
ground_distance);
```

```
        MSP_Message serialize_PX4FLOW_Request();

        void set_PX4FLOW_Handler(class PX4FLOW_Handler * handler);

        MSP_Message serialize_SET_RAW_RC(short c1, short c2, short c3,
short c4, short c5, short c6, short c7, short c8);

        MSP_Message serialize_MSP_STATUS(short cycleTime, short
i2c_errors_count, short sensor, int flag, byte currentSet);

        MSP_Message serialize_MSP_STATUS_Request();

        void set_MSP_STATUS_Handler(class MSP_STATUS_Handler *
handler);

        MSP_Message serialize_DEBUG(byte c1, byte c2, byte c3, byte
c4, byte c5, byte c6, byte c7, byte c8, byte c9, byte c10, byte c11,
byte c12, byte c13, byte c14, byte c15, byte c16);

        MSP_Message serialize_DEBUG_Request();

        void set_DEBUG_Handler(class DEBUG_Handler * handler);

    private:

        class RC_Handler * handlerForRC;

        class ATTITUDE_Handler * handlerForATTITUDE;

        class MOTOR_Handler * handlerForMOTOR;

        class PX4FLOW_Handler * handlerForPX4FLOW;

        class MSP_STATUS_Handler * handlerForMSP_STATUS;

        class DEBUG_Handler * handlerForDEBUG;

};


class RC_Handler {

    public:

        RC_Handler() {}

        virtual void handle_RC(short c1, short c2, short c3, short c4,
short c5, short c6, short c7, short c8, short c9, short c10, short
c11, short c12, short c13, short c14, short c15, short c16, short c17,
short c18){ }

};



class ATTITUDE_Handler {
```

```
    public:

        ATTITUDE_Handler() {}

        virtual void handle_ATTITUDE(short angx, short angy, short
heading){ }

};



class MOTOR_Handler {

    public:

        MOTOR_Handler() {}

        virtual void handle_MOTOR(short c1, short c2, short c3, short
c4, short c5, short c6, short c7, short c8){ }

};



class PX4FLOW_Handler {

    public:

        PX4FLOW_Handler() {}

        virtual void handle_PX4FLOW(short framecount, short
pixel_flow_x_sum, short pixel_flow_y_sum, short flow_comp_m_x, short
flow_comp_m_y, short qual, short gyro_x_rate, short gyro_y_rate, short
gyro_z_rate, byte gyro_range, byte sonar_timestamp, short
ground_distance){ }

};



class MSP_STATUS_Handler {

    public:

        MSP_STATUS_Handler() {}

        virtual void handle_MSP_STATUS(short cycleTime, short
i2c_errors_count, short sensor, int flag, byte currentSet){ }

};



class DEBUG_Handler {

    public:

        DEBUG_Handler() {}
```

```
        virtual void handle_DEBUG(byte c1, byte c2, byte c3, byte c4,
byte c5, byte c6, byte c7, byte c8, byte c9, byte c10, byte c11, byte
c12, byte c13, byte c14, byte c15, byte c16){ }

};
```

4.4 Подтрек «SpaceNet»

Задача:

Необходимо разработать бортовой компьютера спутника навигации позволяющий решать задачу позиционирования аппарата соответствии с поставленной задачей для определения координат корабля или машины.

Камера спутника распознает метку и передает ее координаты по радио. Однако поскольку камера не охватывает весь полигон, то спутник необходимо перемещать. В данном задании вам необходимо собрать систему которая сможет автоматически или по запросу управлять перемещением спутника.

Базовый набор:

- Контроллер на базе Arduino
- Макетная плата, разъем
- Радиомодуль
- Контроллер шагового двигателя
- Набор электронных компонентов

Электрическая схема:

Вариант принципиальной схемы подключения элементов бортового компьютера конструктора спутника. Обратите внимание на схему питания системы.

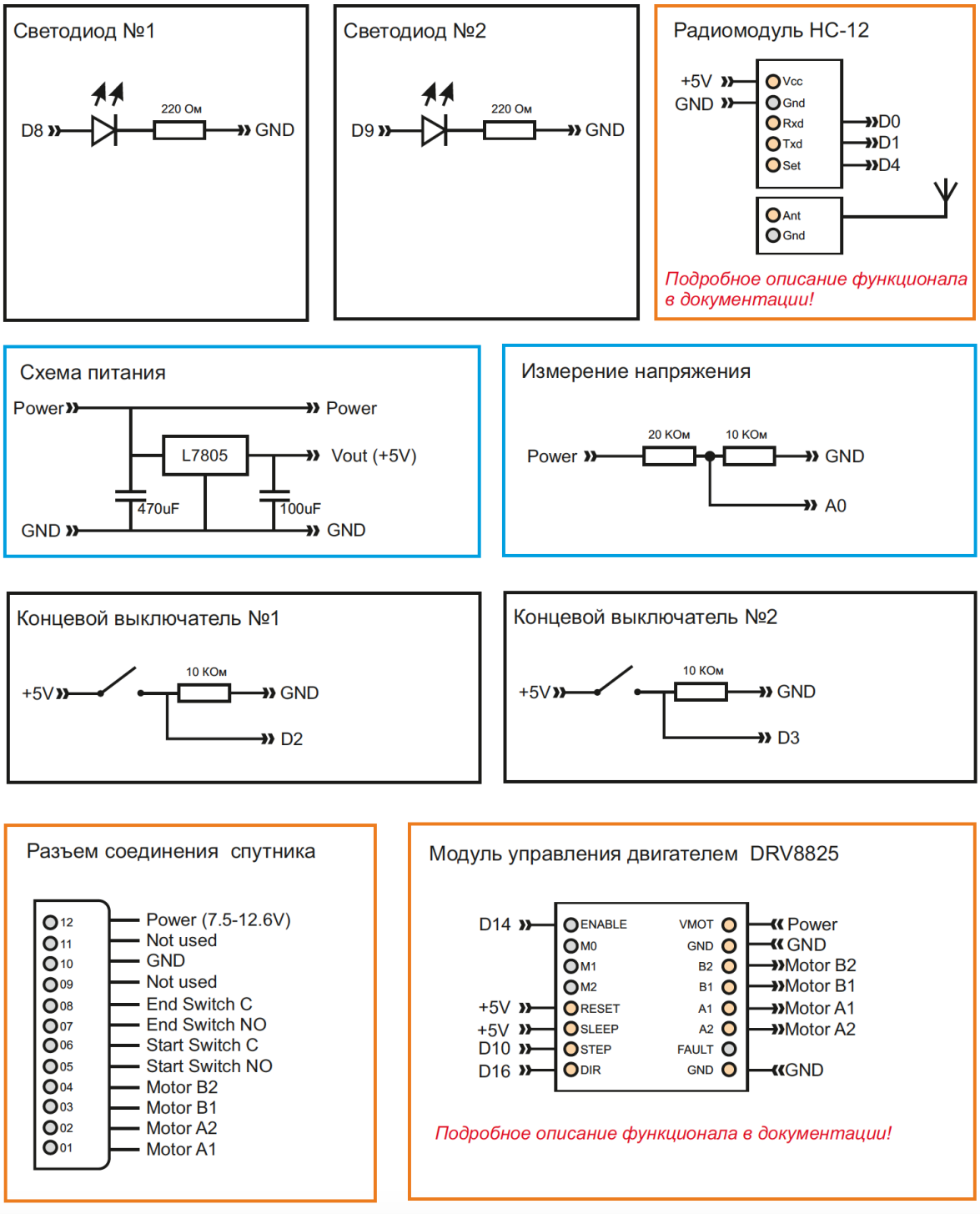


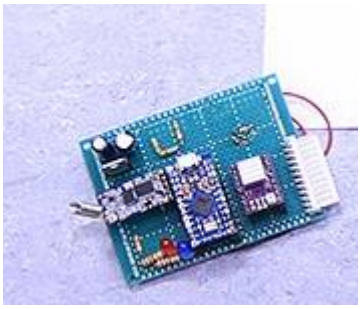
Схема полигона:

Спутник перемещается вдоль полигона над зоной Маринета и Автонета на расстояние около 8 метров. За перемещение спутника вдоль полигона отвечает шаговый двигатель который делает 200 шагов на 1 оборот. На вал установлены шкивы GT2 с 20 зубьями. Смещение на 1 зуб составляет 2 мм.

Баллы и задания:

	Задача	Баллы
1	Собрать бортовой компьютер спутника и пройти тестирование на стенде: 1. Спаять плату 2. Продемонстрировать работу: a. Шагового двигателя b. Радиопередатчика c. Концевого выключателя старта d. Концевого выключателя финиша e. Контроля напряжения	10
2	Запустить проверочную программу спутника предоставленную организаторами (или ее модификацию в случае сборки не по схеме), продемонстрировать работу спутника на полигоне. Проверяется судьей путем посылки управляющих команд на спутник. Перед тестированием на полигоне проводится проверка на стенде.	5
3	Слежение за объектом Написать программу которая следит за меткой. Судья перемещает метку по полигону, спутник следует за меткой, занимая позицию над ней. Предварительно выполнить 1 и 2 задачи	15

Вариант финальной сборки устройства:



Полное решение:

```
#define STEP_PIN 10
#define DIR_PIN 16
#define POWER_ENABLE_PIN 14
#define END_SWITCH1 3
#define END_SWITCH2 2

#define LEFT_DIR 0
#define RIGHT_DIR 1

#define LED1 8
#define LED2 9
```

```
#define STEP_IN_ROTATION 200

const int volt_pin = A0;
float Voltage;
int dir, rotation;
long steps, i;


void setup() {
  // put your setup code here, to run once:
  Serial1.begin(9600);
  pinMode(STEP_PIN, OUTPUT);
  pinMode(DIR_PIN, OUTPUT);
  pinMode(POWER_ENABLE_PIN, OUTPUT);
  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);

  digitalWrite(POWER_ENABLE_PIN, HIGH);
  digitalWrite(LED1, HIGH);
  digitalWrite(LED2, HIGH);
  delay(1000);
  digitalWrite(LED1, LOW);
  digitalWrite(LED2, LOW);
}

void loop() {
  // put your main code here, to run repeatedly:
  if (Serial1.available() > 0) {

    // max int 31072
    rotation = Serial1.readStringUntil('\n').toInt();

    if (rotation > 0) {
      move(RIGHT_DIR, rotation);
    }

    if (rotation < 0) {
      move(LEFT_DIR, abs(rotation));
    }

  }

  Voltage = analogRead(volt_pin);
  Serial.print("Voltage:");
  Serial.println(Voltage*15.00/1023.00);

  Serial.println("---");
}

void move(int dir, unsigned int rotation) {

  digitalWrite(POWER_ENABLE_PIN, LOW);
  digitalWrite(DIR_PIN, dir);

  steps = rotation * STEP_IN_ROTATION;

  doSteps(steps);
```

```
  digitalWrite(POWER_ENABLE_PIN, HIGH);

}

boolean isEndSwitch() {

  if (digitalRead(END_SWITCH1) == 1) {
    Serial1.println("END_SWITCH1 (Start)");
    rollBack(RIGHT_DIR);
    return true;
  }

  if (digitalRead(END_SWITCH2) == 1) {
    Serial1.println("END_SWITCH2 (Finish)");
    rollBack(LEFT_DIR);
    return true;
  }

  return false;
}

boolean isStopInSerial() {

  if (Serial1.available() > 0) {
    if (Serial1.readStringUntil('\n').toInt() == 0) {
      return true;
    }
  }

  return false;

}

void rollBack(int dir) {

  digitalWrite(DIR_PIN, dir);
  for (i = 0; i <= 100; i++) {
    doStep();
  }
}

void doSteps(long steps) {

  for (i = 0; i <= steps; i++) {

    if (isEndSwitch()) {
      return;
    }

    if (isStopInSerial()) {
      return;
    }

    doStep();

  }

  Serial1.println("STOP");
```

```
}

void doStep() {

  digitalWrite(STEP_PIN, HIGH);
  delayMicroseconds(800);
  digitalWrite(STEP_PIN, LOW);
  delayMicroseconds(800);

}#define STEP_PIN 10
#define DIR_PIN 16
#define POWER_ENABLE_PIN 14
#define END_SWITCH1 3
#define END_SWITCH2 2

#define LEFT_DIR 0
#define RIGHT_DIR 1

#define LED1 8
#define LED2 9

#define STEP_IN_ROTATION 200

const int volt_pin = A0;
float Voltage;
int dir, rotation;
long steps, i;


void setup() {
  // put your setup code here, to run once:
  Serial1.begin(9600);
  pinMode(STEP_PIN, OUTPUT);
  pinMode(DIR_PIN, OUTPUT);
  pinMode(POWER_ENABLE_PIN, OUTPUT);
  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);

  digitalWrite(POWER_ENABLE_PIN, HIGH);
  digitalWrite(LED1, HIGH);
  digitalWrite(LED2, HIGH);
  delay(1000);
  digitalWrite(LED1, LOW);
  digitalWrite(LED2, LOW);
}

void loop() {
  // put your main code here, to run repeatedly:
  if (Serial1.available() > 0) {

    // max int 31072
    rotation = Serial1.readStringUntil('\n').toInt();

    if (rotation > 0) {
      move(RIGHT_DIR, rotation);
    }
```

```
    if (rotation < 0) {
      move(LEFT_DIR, abs(rotation));
    }

  }

  Voltage = analogRead(volt_pin);
  Serial.print("Voltage:");
  Serial.println(Voltage*15.00/1023.00);

  Serial.println("---");
}

void move(int dir, unsigned int rotation) {

  digitalWrite(POWER_ENABLE_PIN, LOW);
  digitalWrite(DIR_PIN, dir);

  steps = rotation * STEP_IN_ROTATION;

  doSteps(steps);

  digitalWrite(POWER_ENABLE_PIN, HIGH);

}

boolean isEndSwitch() {

  if (digitalRead(END_SWITCH1) == 1) {
    Serial1.println("END_SWITCH1 (Start)");
    rollBack(RIGHT_DIR);
    return true;
  }

  if (digitalRead(END_SWITCH2) == 1) {
    Serial1.println("END_SWITCH2 (Finish)");
    rollBack(LEFT_DIR);
    return true;
  }

  return false;
}

boolean isStopInSerial() {

  if (Serial1.available() > 0) {
    if (Serial1.readStringUntil('\n').toInt() == 0) {
      return true;
    }
  }

  return false;

}

void rollBack(int dir) {

  digitalWrite(DIR_PIN, dir);
```

```
  for (i = 0; i <= 100; i++) {
    doStep();
  }
}

void doSteps(long steps) {

  for (i = 0; i <= steps; i++) {

    if (isEndSwitch()) {
      return;
    }

    if (isStopInSerial()) {
      return;
    }

    doStep();

  }

  Serial1.println("STOP");

}

void doStep() {

  digitalWrite(STEP_PIN, HIGH);
  delayMicroseconds(800);
  digitalWrite(STEP_PIN, LOW);
  delayMicroseconds(800);

}
```

4.5 Общие задачи АТС

Задание:

Основная задача трека - перемещение в автономном режиме груза через 3 зоны трека: водную, автомобильную и воздушную. Полная задача прохождения трека АТС приносит команде дополнительные баллы.

Баллы и задачи (начисляются последовательно):

№	Задача	Баллы
1	Корабль доплыл до нужного порта, остановился и груз перегружен на машину, машина отъехала от зоны парковки	10
2	Машина доехала до зоны коптера и груз перегружен на коптер, коптер взлетел	20
3	Коптер перенес груз	20
ИТОГО	Последовательно и без вмешательства перевезти груз по водному, автомобильному и воздушному трекам.	50

* Задание выполняется непрерывно, старт выполняется с зоны корабля. Применяются штрафы согласно условиям треков.