Приложение А. Текст программы

АННОТАЦИЯ

В данном программном документе приведен текст приложения с предиктивной коррекцией ошибок управления (на примере ООО «Центр инновационных разработок ВАО»).

В разделе «Текст программы» указано назначение программы, краткая характеристика области применения программы, описание модулей и их программный код.

СОДЕРЖАНИЕ

1. ТЕКСТ ПРОГРАММЫ

1.1. Наименование программы

Наименование – Встроенное приложение с предиктивной коррекцией ошибок управления.

1.2. Область применения программы

Программа должна эксплуатироваться в составе программно-аппаратного комплекса в виде платформы-носителя с универсальным интерфейсом связи «MasterLink». Конечными пользователями программы должны являться сотрудники с допуском работы на промышленном оборудовании с автоматическим управлением подвижными частями.

1.3. Модули

Таблица 1 - Модули.

№	Название модуля	Описание модуля	Размер модуля	Кол-во строк
1	PlatformMain.cpp	Модуль основной программы	1,2 кб	36
2	Platform.h	Заголовочный файл библиотеки Platform	4,6 кб	147
3	Platform.cpp	Модуль логики библиотеки Platform	23,9 кб	628
4	Arduino.h	Заголовочный файл библиотеки Arduino	7,2 кб	260
5	Display.cpp	Модуль программы полезной нагрузки «Дисплей»	3,2 кб	116

1.4. Код программы

1.4.1. PlatformMain.cpp

```
#include "Platform.h"

Platform platform;

void setup() {
  pinMode(13, OUTPUT); //Debug signal
  Serial.begin(115200); //Debug or platform's load
  Serial1.begin(9600); //GPS
  platform.begin("testPlatf", "8tegqHu6VZ");
  platform.GPIOSetup(GPIO_DIGITALOUT, GPIO_DIGITALOUT, GPIO_DIGITALOUT,
GPIO_DIGITALOUT);
  platform.initUARTControlData(platform);
  platform.initMPU();
}

void loop() {
  while (1) { //Speed-up bug
    //PORTB |= (1 << 7); //13 test square generator
    //PORTB &= ~ (1 << 7); //13

    if (millis() % 50 == 0) {
      //platform.sendUARTControlData("^:asd;\r\n");
      platform.getGPSData(&Serial1);
      platform.getMPUData();
    }

    //  if (Serial.available() > 0) { //Segment for test bridge between PC and platform's load
    //    platform.sendUARTCommandData("^:" + Serial.readString() + ";");
    //  }

    //platform.startBench();
    //delay(500);
    //platform.getGPSData(&Serial1);
    //platform.stopBench(&Serial);
  }
}
```

1.4.2. Platform.h

```
#pragma once
#include <Arduino.h>
#include <avr/interrupt.h>
#include <Wire.h>
#define _LIB_VERSION        1.0

#define DEBUGGYRO           false
#define DEBUGACC            false
#define DEBUGUART           false
#define DEBUGGPS            false

#define MPU6050_ADDRESS     0x68
```

```
#define BACKWARD          0      // Move backward
#define FORWARD           1      // Move forward
#define LEFT              2      // Move counterclock-wise
#define RIGHT             3      // Move counterclock
#define FORWARDLEFT       4      // Move forward and left
#define FORWARDRIGHT      5      // Move forward and right
#define BACKWARDLEFT      6      // Move backward and left
#define BACKWARDRIGHT     7      // Move backward and right

#define BRAKE             1      // Value for rapid braking
#define STOP              0      // Value for inertional braking
#define FAST              0      // Value for rapid acceleration
#define SLOW              1      // Value for soft acceleration

#define STATUS_STOP       0      // Stop, command processing is discontinued
#define STATUS_WORK       1      // Work, exchange of commands
#define STATUS_SHUTDOWN   2      // Ready to Shut Down
#define STATUS_ECO        3      // Energy saving mode
#define STATUS_EMODE      4      // Emergency mode
#define STATUS_ERROR      5      // Unexpected system error
#define STATUS_EXEPTION   6      // Work, have problems

#define GPIO_OFF          0      // GPIO off
#define GPIO_DIGITALIN    1      // GPIO as digital input
#define GPIO_DIGITALOUT   2      // GPIO as digital output
#define GPIO_ANALOGIN     3      // GPIO as analog input

struct DataIncome {                              // Structure of data coming from PC to UART
  char move;
  uint8_t speed;
  char value;
  uint8_t azimuthloc;
  uint8_t gpio1 = 0;
  uint8_t gpio2 = 0;
  uint8_t gpio3 = 0;
  uint8_t gpio4 = 0;
  uint8_t systemstatus = 0;
  String data;
};

struct DataOutcome {                             // Data structure from UART to PC
  char move;
  uint8_t speed;
  char value;
  uint16_t lcurr;
  uint16_t rcurr;
  float accx;
  float accy;
  float accz;
  float gyrox;
  float gyroy;
  float gyroz;
  float magx;
  float magy;
  float magz;
  String lan;
  String lon;
  float vbat;
  uint8_t systemstatus = 0;
  uint16_t extid = 0;
  uint8_t extstatus = 0;
};

struct MainParameters {        // Data structure of platform parameters
  uint8_t systemstatus = 0;
  uint16_t extid = 0;
  uint8_t extstatus = 0;

  String GPSTimestamp ="";
  String GPSLatitude = "0.000000";
  String GPSLongitude = "0.000000";
};

class Platform {   // class Platform
  public:
    DataIncome controlDataIn;
    DataOutcome controlDataOut;
    MainParameters mainParameters;

    //GPIO mode
    uint8_t GPIO1 = 0;
    uint8_t GPIO2 = 0;
    uint8_t GPIO3 = 0;
    uint8_t GPIO4 = 0;
    //MPU6050 sensor
    volatile float AccX, AccY, AccZ;
    volatile float GyroX, GyroY, GyroZ;
    volatile float AccErrorX, AccErrorY, GyroErrorX, GyroErrorY, GyroErrorZ;
    volatile float Temperature;
    volatile int MPU_Calib_Counter = 0;
    volatile float AccDevider, GyroDevider = 0;

    Platform();
    void begin(String name, String key);

    //Movements section
    void makeMove(uint8_t direction, uint8_t speed, uint8_t acceleration);
    void brake(uint8_t mode);

    //Telemetry section
    bool initUARTControlData(Platform platform, int baudrate);
    bool initUARTControlData(Platform platform);
    void getUARTControlData(void);
    void sendUARTControlData(String outgoingDataString);
    bool getGPSData(Stream* _serial);
    void initMPU();
    void getMPUData();

    //MasterLink section
    void GPIOSetup(uint8_t GPIO_1, uint8_t GPIO_2, uint8_t GPIO_3, uint8_t GPIO_4);

    //Another useful functions
    void startBench();
    void stopBench(Stream* _serial);
    float convertRawCoordinatesToDegrees(float RawDegrees);
    void I2Cread(uint8_t Address, uint8_t Register, uint8_t Nbytes, uint8_t* Data);
    void I2CwriteByte(uint8_t Address, uint8_t Register, uint8_t Data);

  private:
    String PlatformKey = "";                // Platform's private key
    String PlatformName = "";               // Platform's name

    //Move UART command section
    String stringUARTCommand = "";          // Variable of collection of accepted command characters per line
    volatile bool startedUARTCommandRecieve; // Variable odf uart command data recieve begin
    volatile uint8_t indexUARTCommand = 0;   // Index of accepted command mode argument
```

```
    //Load UART command section
    String stringUARTLoad = "";               // Variable of collecting accepted platform load symbols per string
    volatile bool startedUARTLoadRecieve;     // Platform load data start variable by uart
};
```

1.4.3. Platform.cpp

```
#include "Platform.h"

#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))

Platform ptf; // Class instance call

Platform::Platform() {} // Class constructor

void Platform::begin(String name, String key) {
  PlatformName = name;
  PlatformKey = key;



  pinMode(7, OUTPUT);     // Motor key A, 7
  pinMode(4, OUTPUT);     // Motor key A, 4
  pinMode(8, OUTPUT);     // Motor key B, 8
  pinMode(9, OUTPUT);     // Motor key B, 9
  pinMode(5, OUTPUT);     // Motor PWM pin, 5
  pinMode(6, OUTPUT);     // Motor PWM pin, 6
  pinMode(A2, INPUT);     // Current sensor pin, A2
  pinMode(A3, INPUT);     // Current sensor pin, A3
  pinMode(A7, INPUT);     // Voltage sensor pin, A7

  pinMode(52, OUTPUT);    // GPIO1 pin
  pinMode(50, OUTPUT);    // GPIO2 pin
  pinMode(51, OUTPUT);    // GPIO3 pin
  pinMode(53, OUTPUT);    // GPIO4 pin

  sbi(TCCR3A, COM3A1);    // PWM, 5
  sbi(TCCR4A, COM4A1);    // PWM, 6

  mainParameters.systemstatus = STATUS_WORK;
}

void Platform::makeMove(uint8_t direction, uint8_t speed, uint8_t acceleration) {
```

```
uint8_t dividerForRightMotor = 0;
uint8_t dividerForLeftMotor = 0;

PORTH &= ~ (1 << 4); //7, LOW A
PORTG &= ~ (1 << 5); //4, LOW A
PORTH &= ~ (1 << 5); //8, LOW B
PORTH &= ~ (1 << 6); //9, LOW B

switch (direction) {
  case 0:
    PORTH |= (1 << 4); //7, HIGH A
    PORTH |= (1 << 6); //9, HIGH B
    break;
  case 1:
    PORTG |= (1 << 5); //4, HIGH A
    PORTH |= (1 << 5); //8, HIGH B
    break;
  case 2:
    PORTH &= ~ (1 << 4); //7, LOW A
    PORTH |= (1 << 5); //8, HIGH B

    PORTG &= ~ (1 << 5); //4, LOW A
    PORTH |= (1 << 6); //9, HIGH B
    break;
  case 3:
    PORTH |= (1 << 4); //7, HIGH A
    PORTH &= ~ (1 << 5); //8, LOW B

    PORTG |= (1 << 5); //4, HIGH A
    PORTH &= ~ (1 << 6); //9, LOW B
    break;
  case 4:
    PORTG |= (1 << 5); //4, HIGH A
    PORTH |= (1 << 5); //8, HIGH B

    dividerForRightMotor = 5;
    dividerForLeftMotor = 0; //Decrease left speed
    break;
  case 5:
    PORTG |= (1 << 5); //4, HIGH A
    PORTH |= (1 << 5); //8, HIGH B

    dividerForRightMotor = 0; //Decrease right speed
```

```
      dividerForLeftMotor = 5;
      break;
    case 6:
      PORTH |= (1 << 4); //7, HIGH A
      PORTH |= (1 << 6); //9, HIGH B

      dividerForRightMotor = 5;
      dividerForLeftMotor = 0; //Decrease left speed
      break;
    case 7:
      PORTH |= (1 << 4); //7, HIGH A
      PORTH |= (1 << 6); //9, HIGH B

      dividerForRightMotor = 0; //Decrease right speed
      dividerForLeftMotor = 5;
      break;
  }

  /*if(acceleration == 1) {
              for(uint16_t i = 0; i <= map(speed, 0, 100, 0, 255); i++) { //Не работает. работает. да...
                            analogWrite(pwmpin[0], i);
                            analogWrite(pwmpin[1], i);
                            delay(10);
              }
    }
    else {*/

  OCR3A = map(speed<5?speed:speed-dividerForLeftMotor, 0, 100, 0, 255); // set pwm duty
  OCR4A = map(speed<5?speed:speed-dividerForRightMotor, 0, 100, 0, 255);

  //}
}

void Platform::brake(uint8_t mode) {
  if (mode == 1) {
    OCR3A = 0; // set pwm duty
    OCR4A = 0;

    //Rapid braking, short circuit motor
    PORTH |= (1 << 4); //7, HIGH
    PORTG |= (1 << 5); //4, HIGH
    PORTH |= (1 << 5); //8, HIGH
    PORTH |= (1 << 6); //9, HIGH
    delay(50);

    //Return keys to low state
    PORTH &= ~ (1 << 4); //7, LOW
    PORTG &= ~ (1 << 5); //4, LOW
    PORTH &= ~ (1 << 5); //8, LOW
    PORTH &= ~ (1 << 6); //9, LOW
  }
  else {
    OCR3A = 0; // set pwm duty
    OCR4A = 0;

    //Soft inertional braking
    PORTH &= ~ (1 << 4); //7, LOW
    PORTG &= ~ (1 << 5); //4, LOW
    PORTH &= ~ (1 << 5); //8, LOW
    PORTH &= ~ (1 << 6); //9, LOW
  }
}

//Telemetry section

bool Platform::initUARTControlData(Platform platform, int baudrate) {
  UCSR2A = 1 << U2X1; //UCSR2A = 1 << U2X1 for 115200
  // assign the baud_setting, a.k.a. ubrr (USART Baud Rate Register)
  /* Set baud rate */
  UBRR2H = baudrate >> 8;
  UBRR2L = baudrate;

  //Permission to receive and transmit via USART, interrupts on arrival and on devastation
  UCSR2B = (1 << RXCIE2) | (1 << TXCIE2) | (1 << RXEN2) | (1 << TXEN2);
  UCSR2C = (1 << UCSZ21) | (1 << UCSZ20); //Word's size 8 bits
  sei();

  ptf = platform;
  return true;
}

bool Platform::initUARTControlData(Platform platform) {
  UCSR2A = 1 << U2X1;
  // assign the baud_setting, a.k.a. ubrr (USART Baud Rate Register)
  /* Set baud rate */
  UBRR2H = 34 >> 8; //Value '34' for 57600 baudrate
  UBRR2L = 34;

  //Permission to receive and transmit via USART, interrupts on arrival and on devastation
  UCSR2B = (1 << RXCIE2) | (1 << TXCIE2) | (1 << RXEN2) | (1 << TXEN2);
  UCSR2C = (1 << UCSZ21) | (1 << UCSZ20); //Word's size 8 bits
  sei();

  ptf = platform;
  return true;
}



ISR(USART2_RX_vect) { //ISR UART2 handler
  if(ptf.mainParameters.systemstatus != STATUS_EMODE) ptf.getUARTControlData();
}

void Platform::getUARTControlData(void) {
  while ( !(UCSR2A & (1 << RXC2)) );
  char incomingByte = UDR2; // Read income char

  //------------------------------------------------Who am I section------------------------------------------------
  if (incomingByte == '@' && !startedUARTCommandRecieve && !startedUARTLoadRecieve) {
    sendUARTControlData("@:"+PlatformName+","+PlatformKey+";");
  }

  //------------------------------------------------Load UART command section-------------------------------------------
-------
  if (incomingByte == '*') {
    startedUARTLoadRecieve = true;
    stringUARTLoad = "";
  }
  if (incomingByte != ';' && startedUARTLoadRecieve) stringUARTLoad += incomingByte;
  else {
    stringUARTLoad += ";";

    for (uint32_t i = 0; i <= strlen(stringUARTLoad.c_str()); ++i) { //UART0 transmit
      /* Wait for empty transmit buffer */
      while ( !( UCSR0A & (1 << UDRE0)) );
      /* Put data into buffer, sends the data */
      UDR0 = stringUARTLoad[i];
    }
    startedUARTLoadRecieve = false;
    stringUARTLoad = "";
  }

  //------------------------------------------------Move UART command section-----------------------------------------
--------
  if (incomingByte != ',' && incomingByte != ';' && startedUARTCommandRecieve &&
!startedUARTLoadRecieve) {  // if it isn't space and end
    stringUARTCommand += incomingByte;                                      // Add to sting
  } else {                                                                  // If it's a space or ;
      switch (indexUARTCommand) {
        case 0:
          controlDataIn.move = stringUARTCommand[1];
          break;
        case 1:
          controlDataIn.speed = stringUARTCommand.toInt();
          break;
        case 2:
          controlDataIn.value = stringUARTCommand[0];
          break;
        case 3:
          controlDataIn.azimuthloc = stringUARTCommand.toInt();
          break;
        case 4:
          controlDataIn.gpio1 = stringUARTCommand.toFloat();
          if(GPIO1 == GPIO_DIGITALOUT) digitalWrite(52, stringUARTCommand.toFloat());
          break;
        case 5:
          controlDataIn.gpio2 = stringUARTCommand.toFloat();
          if(GPIO2 == GPIO_DIGITALOUT) digitalWrite(50, stringUARTCommand.toFloat());
          break;
        case 6:
          controlDataIn.gpio3 = stringUARTCommand.toFloat();
          if(GPIO3 == GPIO_DIGITALOUT) digitalWrite(51, stringUARTCommand.toFloat());
          break;
        case 7:
          controlDataIn.gpio4 = stringUARTCommand.toFloat();
          if(GPIO4 == GPIO_DIGITALOUT) digitalWrite(53, stringUARTCommand.toFloat());
          break;
        case 8:
          controlDataIn.systemstatus = stringUARTCommand.toInt();
          ptf.mainParameters.systemstatus = controlDataIn.systemstatus;
          break;
        case 9:
          controlDataIn.data = stringUARTCommand;
          break;
      }
    stringUARTCommand = "";                    // Clear string
    indexUARTCommand++;                        // Select next parsing section of array
  }
  if (incomingByte == '%') {
    startedUARTCommandRecieve = true;
    indexUARTCommand = 0;
    stringUARTCommand = "";
  }
  if (incomingByte == ';' && startedUARTCommandRecieve) {
    startedUARTCommandRecieve = false;

    //Заполняем структуру и передаем её
    if(mainParameters.systemstatus != STATUS_STOP && mainParameters.systemstatus != STATUS_EMODE) {
      controlDataOut.move = controlDataIn.move;
      controlDataOut.speed = controlDataIn.speed;
      controlDataOut.value = controlDataIn.value;
    }
    controlDataOut.lcurr = analogRead(A3) * 0.038;   //Current in Amps
    controlDataOut.rcurr = analogRead(A2) * 0.038;
    // controlDataOut.accx = AccX;
    // controlDataOut.accy = AccY;
    // controlDataOut.accz = AccZ;
    // controlDataOut.gyrox = GyroX;
    // controlDataOut.gyroy = GyroY;
    // controlDataOut.gyroz = GyroZ;
    controlDataOut.magx = 0;
    controlDataOut.magy = 0;
    controlDataOut.magz = 0;
    controlDataOut.lan = mainParameters.GPSLatitude;
    controlDataOut.lon = mainParameters.GPSLongitude;
    controlDataOut.vbat = ((analogRead(A7)* 5.0) / 1024.0)/0.337;
    controlDataOut.systemstatus = mainParameters.systemstatus;
    controlDataOut.extid = mainParameters.extid;
    controlDataOut.extstatus = mainParameters.systemstatus;

    //Serial.println(ptf.controlDataOut.gyrox);

    String outgoingDataString = "&:" + String(controlDataOut.move) + "," + String(controlDataOut.speed) +
"," + String(controlDataOut.value) + "," + String(controlDataOut.lcurr) + "," + String(controlDataOut.rcurr)
+ "," + String(controlDataOut.accx) + "," + String(controlDataOut.accy) + "," + String(controlDataOut.accz)
+ "," + String(controlDataOut.gyrox) + "," + String(controlDataOut.gyroy) + "," +
String(controlDataOut.gyroz) + "," + String(controlDataOut.magx) + "," + String(controlDataOut.magy) +
"," + String(controlDataOut.magz) + "," + controlDataOut.lan + "," + controlDataOut.lon + "," +
String(controlDataOut.vbat) + "," + String(controlDataOut.systemstatus) + "," + String(controlDataOut.extid)
+ "," + String(controlDataOut.extstatus) + ";\r\n";
  //String outgoingDataString = "&:" +PlatformName+"," + String(controlDataOut.move) + "," +
String(controlDataOut.speed) + "," + String(controlDataOut.value) + "," + String(controlDataOut.lcurr) + ","
+ String(controlDataOut.rcurr) + "," + String(controlDataOut.accx) + "," + String(controlDataOut.accy) + ","
+ String(controlDataOut.accz) + "," + String(controlDataOut.gyrox) + "," + String(controlDataOut.gyroy) +
"," + String(controlDataOut.gyroz) + "," + String(controlDataOut.magx) + "," +
String(controlDataOut.magy) + "," + String(controlDataOut.magz) + "," + controlDataOut.lan + "," +
controlDataOut.lon + "," + String(controlDataOut.vbat) + "," + String(controlDataOut.extid) + "," +
String(controlDataOut.extstatus) + ";\r\n";

  sendUARTControlData(outgoingDataString);

  if(ptf.mainParameters.systemstatus != STATUS_STOP && ptf.mainParameters.systemstatus !=
STATUS_EMODE) {
    switch (controlDataIn.move) {
      case 'f':
        makeMove(FORWARD, controlDataIn.speed, (controlDataIn.value == 'f') ? FAST : SLOW);
        break;
      case 'b':
        makeMove(BACKWARD, controlDataIn.speed, (controlDataIn.value == 'f') ? FAST : SLOW);
        break;
      case 'l':
        makeMove(LEFT, controlDataIn.speed, (controlDataIn.value == 'f') ? FAST : SLOW);
        break;
      case 'r':
        makeMove(RIGHT, controlDataIn.speed, (controlDataIn.value == 'f') ? FAST : SLOW);
        break;
      case 'a':
        makeMove(FORWARDLEFT, controlDataIn.speed, (controlDataIn.value == 'f') ? FAST : SLOW);
        break;
      case 'c':
        makeMove(FORWARDRIGHT, controlDataIn.speed, (controlDataIn.value == 'f') ? FAST : SLOW);
        break;
      case 'd':
        makeMove(BACKWARDLEFT, controlDataIn.speed, (controlDataIn.value == 'f') ? FAST : SLOW);
        break;
      case 'e':
        makeMove(BACKWARDRIGHT, controlDataIn.speed, (controlDataIn.value == 'f') ? FAST : SLOW);
        break;
      case 's':
        brake(STOP);
        break;
    }
  }
  else brake(BRAKE);
}
}

void Platform::sendUARTControlData(String outgoingDataString)
{
for (uint32_t i = 0; i <= strlen(outgoingDataString.c_str()); ++i) {
  /* Wait for empty transmit buffer */
  while ( !( UCSR2A & (1 << UDRE2)) );
  /* Put data into buffer, sends the data */
  UDR2 = outgoingDataString[i];
}
}

bool Platform::getGPSData(Stream* _serial) {
String stringGPS = "";
if (_serial->available() > 0) {
  stringGPS = _serial->readStringUntil(13); //NMEA data ends with 'return' character, which is ascii(13)
  stringGPS.trim();                      // they say NMEA data starts with "$", but the Arduino doesn't think so.
  //Serial.println(stringGPS);         //All the raw sentences will be sent to monitor, if you want them, maybe
to see the labels and data order.

  //Start Parsing by finding data, put it in a string of character array, then removing it, leaving the rest of thes
sentence for the next 'find'
  if (stringGPS.startsWith("$GPGLL") || stringGPS.startsWith("$GLGLL") ||
stringGPS.startsWith("$GAGLL") || stringGPS.startsWith("$BDGLL") || stringGPS.startsWith("$GQGLL") ||
stringGPS.startsWith("$GNGLL")) { //I picked this sentence, you can pick any of the other labels and
rearrange/add sections as needed.
    //Serial.println(stringGPS);     // display raw GLL data in Serial Monitor
    // mine looks like this: "$GPGLL,4053.16598,N,10458.93997,E,224431.00,A,D*7D"

    //This section gets repeated for each delimeted bit of data by looking for the commas
    //Find Lattitude is first in GLL sentence, other senetences have data in different order
    int Pos = stringGPS.indexOf(','); //look for comma delimetrer
    stringGPS.remove(0, Pos + 1); // Remove Pos+1 characters starting at index=0, this one strips off
"$GPGLL" in my sentence
```

```
  Pos = stringGPS.indexOf(','); //looks for next comma delimetrer, which is now the first comma because I removed the first segment
  char Lat[Pos];            //declare character array Lat with a size of the dbit of data
  for (int i = 0; i <= Pos - 1; i++) { // load charcters into array
    Lat[i] = stringGPS.charAt(i);
  }
  //Serial.print(Lat);          // display raw latitude data in Serial Monitor, I'll use Lat again in a few lines for converting
  //repeating with a different char array variable
  //Get Lattitude North or South
  stringGPS.remove(0, Pos + 1);
  Pos = stringGPS.indexOf(',');
  char LatSide[Pos];           //declare different variable name
  for (int i = 0; i <= Pos - 1; i++) {
    LatSide[i] = stringGPS.charAt(i); //fill the array
    //Serial.println(LatSide[i]);       //display N or S
  }

  //convert the variable array Lat to degrees Google can use
  float LatAsFloat = atof (Lat);            //atof converts the char array to a float type
  float LatInDeg;
  if (LatSide[0] == char(78)) {     //char(69) is decimal for the letter "N" in ascii chart
    LatInDeg = convertRawCoordinatesToDegrees(LatAsFloat);  //call the conversion funcion (see below)
  }
  if (LatSide[0] == char(83)) {     //char(69) is decimal for the letter "S" in ascii chart
    LatInDeg = -( convertRawCoordinatesToDegrees(LatAsFloat));  //call the conversion funcion (see below)
  }
  if(LatInDeg > 0 && String(LatInDeg, 8) != "") ptf.mainParameters.GPSLatitude = String(LatInDeg, 8); //TEMP SOLUTION
  //Serial.println(LatInDeg, 15); //display value Google can use in Serial Monitor, set decimal point value high
  //repeating with a different char array variable
  //Get Longitude
  stringGPS.remove(0, Pos + 1);
  Pos = stringGPS.indexOf(',');
  char Longit[Pos];             //declare different variable name
  for (int i = 0; i <= Pos - 1; i++) {
    Longit[i] = stringGPS.charAt(i);    //fill the array
  }
  //Serial.print(Longit);      //display raw longitude data in Serial Monitor
  //repeating with a different char array variable
  //Get Longitude East or West
```

```
      stringGPS.remove(0, Pos + 1);
      Pos = stringGPS.indexOf(',');
      char LongitSide[Pos];        //declare different variable name
      for (int i = 0; i <= Pos - 1; i++) {
        LongitSide[i] = stringGPS.charAt(i);    //fill the array
        //Serial.println(LongitSide[i]);       //display raw longitude data in Serial Monitor
      }
      //convert to degrees Google can use
      float LongitAsFloat = atof (Longit);    //atof converts the char array to a float type
      float LongInDeg;
      if (LongitSide[0] == char(69)) {     //char(69) is decimal for the letter "E" in ascii chart
        LongInDeg = convertRawCoordinatesToDegrees(LongitAsFloat); //call the conversion funcion (see
below
      }
      if (LongitSide[0] == char(87)) {      //char(87) is decimal for the letter "W" in ascii chart
        LongInDeg = -(convertRawCoordinatesToDegrees(LongitAsFloat)); //call the conversion funcion (see
below
      }
      if(LongInDeg > 0 && String(LongInDeg, 8) != "") ptf.mainParameters.GPSLongitude =
String(LongInDeg, 8); //TEMP SOLUTION
      //Serial.println(LongInDeg, 15); //display value Google can use in Serial Monitor, set decimal point value
high
      //repeating with a different char array variable
      //Get TimeStamp - GMT
      stringGPS.remove(0, Pos + 1);
      Pos = stringGPS.indexOf(',');
      char TimeStamp[Pos];         //declare different variable name
      for (int i = 0; i <= Pos - 1; i++) {
        TimeStamp[i] = stringGPS.charAt(i);      //fill the array
      }
      ptf.mainParameters.GPSTimestamp = TimeStamp; //TEMP SOLUTION
      //Serial.print(TimeStamp);   //display raw longitude data in Serial Monitor, GMT
      //Serial.println(String(LongInDeg, 8));
    }
  }
  return true;
}

void Platform::initMPU() {
  Wire.begin();
  Wire.setClock(400000);

  I2CwriteByte(MPU6050_ADDRESS, 29, 0x06);// Set accelerometers low pass filter at 5Hz !
  I2CwriteByte(MPU6050_ADDRESS, 26, 0x06); // Set gyroscope low pass filter at 5Hz !

  // Configure gyroscope range
  I2CwriteByte(MPU6050_ADDRESS, 27, 0x6B); GyroDevider = 131; //GYRO_FULL_SCALE_250_DPS !
  //I2CwriteByte(MPU6050_ADDRESS, 27, 0x08); GyroDevider = 65.5; //GYRO_FULL_SCALE_500_DPS
  //I2CwriteByte(MPU6050_ADDRESS, 27, 0x10); GyroDevider = 32.8;
//GYRO_FULL_SCALE_1000_DPS
  // I2CwriteByte(MPU6050_ADDRESS, 27, 0x18); GyroDevider = 16.4;
//GYRO_FULL_SCALE_2000_DPS

  // Configure accelerometers range
  I2CwriteByte(MPU6050_ADDRESS, 28, 0x00); AccDevider = 16384; //ACC_FULL_SCALE_2_G !
  //I2CwriteByte(MPU6050_ADDRESS, 28, 0x08); AccDevider = 8192; //ACC_FULL_SCALE_4_G
  //I2CwriteByte(MPU6050_ADDRESS, 28, 0x10); AccDevider = 4096; //ACC_FULL_SCALE_8_G
  //I2CwriteByte(MPU6050_ADDRESS, 28, 0x18); AccDevider = 2048; //ACC_FULL_SCALE_16_G



  while (MPU_Calib_Counter < 200) {
    uint8_t Buf[14];
    I2Cread(MPU6050_ADDRESS, 0x3B, 14, Buf);

    //Get values from sensor
    GyroX = -(Buf[0] << 8 | Buf[1]);
    GyroY = -(Buf[2] << 8 | Buf[3]);
    GyroZ = Buf[4] << 8 | Buf[5];

    // Sum all readings
    GyroErrorX = GyroErrorX + (GyroX / GyroDevider);
    GyroErrorY = GyroErrorY + (GyroY / GyroDevider);
    GyroErrorZ = GyroErrorZ + (GyroZ / GyroDevider);
    MPU_Calib_Counter++;
  }

  //Divide the sum by 200 to get the error value
  GyroErrorX = GyroErrorX / 200;
  GyroErrorY = GyroErrorY / 200;
  GyroErrorZ = GyroErrorZ / 200;
  MPU_Calib_Counter = 0;

  while (MPU_Calib_Counter < 200) {
    uint8_t Buf[14];
    I2Cread(MPU6050_ADDRESS, 0x3B, 14, Buf);
    //Get values from sensor
    AccX = (Buf[8] << 8 | Buf[9]) / AccDevider;
    AccY = (Buf[10] << 8 | Buf[11]) / AccDevider;
    AccZ = (Buf[12] << 8 | Buf[13]) / AccDevider;

    // Sum all readings
    AccErrorX = AccErrorX + ((atan((AccY) / sqrt(pow((AccX), 2) + pow((AccZ), 2))) * 180 / PI));
    AccErrorY = AccErrorY + ((atan(-1 * (AccX) / sqrt(pow((AccY), 2) + pow((AccZ), 2))) * 180 / PI));
    MPU_Calib_Counter++;
  }

  //Divide the sum by 200 to get the error value
  AccErrorX = AccErrorX / 200;
  AccErrorY = AccErrorY / 200;
  MPU_Calib_Counter = 0;

#if DEBUGGYRO || DEBUGACC
  Serial.print(F("AccErrorX: "));
  Serial.println(AccErrorX);
  Serial.print(F("AccErrorY: "));
  Serial.println(AccErrorY);
  Serial.print(F("GyroErrorX: "));
  Serial.println(GyroErrorX);
  Serial.print(F("GyroErrorY: "));
  Serial.println(GyroErrorY);
  Serial.print(F("GyroErrorZ: "));
  Serial.println(GyroErrorZ);
#endif
}

void Platform::getMPUData() {
  uint8_t Buf[14];

  I2Cread(MPU6050_ADDRESS, 0x3B, 14, Buf); // Read accelerometer and gyroscope

  //Gyroscope
  GyroX = (Buf[0] << 8 | Buf[1]) / GyroDevider;
  GyroY = (Buf[2] << 8 | Buf[3]) / GyroDevider;
  GyroZ = (Buf[4] << 8 | Buf[5]) / GyroDevider;

  // Correct the outputs with the calculated error values
  GyroX = GyroX + abs(GyroErrorX); // GyroErrorX ~(-0.56)
  GyroY = GyroY + abs(GyroErrorY); // GyroErrorY ~(2)
  GyroZ = GyroZ + abs(GyroErrorZ); // GyroErrorZ ~ (-0.8)

  //Temperature
  Temperature = (Buf[6] << 8 | Buf[7]) / 340.0 + 36.53;

  // Accelerometer
  AccX = (Buf[8] << 8 | Buf[9]) / AccDevider;
  AccY = (Buf[10] << 8 | Buf[11]) / AccDevider;
  AccZ = (Buf[12] << 8 | Buf[13]) / AccDevider;

  // Display values
  ptf.controlDataOut.accx = AccX;
  ptf.controlDataOut.accy = AccY;
  ptf.controlDataOut.accz = AccZ;
  ptf.controlDataOut.gyrox = GyroX;
  ptf.controlDataOut.gyroy = GyroY;
  ptf.controlDataOut.gyroz = GyroZ;
  //Serial.println(ptf.controlDataOut.gyrox);

  // Gyroscope
#if DEBUGGYRO
  Serial.print(F("GyroX: "));
  Serial.println((int)GyroX, DEC);
  Serial.print(F("GyroY: "));
  Serial.println((int)GyroY, DEC);
  Serial.print(F("GyroZ: "));
  Serial.println((int)GyroZ, DEC);
  Serial.println((int)Temperature, DEC);
#endif

  // Accelerometer
#if DEBUGACC
  Serial.print(F("AccX: "));
    Serial.println(AccX, DEC);
    Serial.print(F("AccY: "));
    Serial.println(AccY, DEC);
    Serial.print(F("AccZ: "));
    Serial.println (AccZ, DEC);
#endif

}

//MasterLink section
```

```
void Platform::GPIOSetup(uint8_t GPIO_1, uint8_t GPIO_2, uint8_t GPIO_3, uint8_t GPIO_4) {
  GPIO1 = GPIO_1;
  GPIO2 = GPIO_2;
  GPIO3 = GPIO_3;
  GPIO4 = GPIO_4;

  if(GPIO_1 == GPIO_OFF || GPIO_1 == GPIO_DIGITALOUT) pinMode(52, OUTPUT);
  else pinMode(52, INPUT);

  if(GPIO_2 == GPIO_OFF || GPIO_2 == GPIO_DIGITALOUT) pinMode(50, OUTPUT);
  else pinMode(50, INPUT);

  if(GPIO_3 == GPIO_OFF || GPIO_3 == GPIO_DIGITALOUT) pinMode(51, OUTPUT);
  else pinMode(51, INPUT);

  if(GPIO_4 == GPIO_OFF || GPIO_4 == GPIO_DIGITALOUT) pinMode(53, OUTPUT);
  else pinMode(53, INPUT);
}

//Another useful functions
void Platform::startBench() {
  TCCR1A = 0x00;          // Turn off
  TCCR1B = 0x00;          // Turn off
  TCNT1 = 0x00;           // Reset counter
  TCCR1B = 0x01;          // Start timer
}

void Platform::stopBench(Stream* _serial) {
  TCCR1B = 0x00;          // Stop timer
  uint32_t count = TCNT1 - 2; // Minus 2 ticks on actions

  _serial->print("ticks: ");
  _serial->print(count);
  _serial->print("  ");
  _serial->print("time (us): ");
  _serial->println(count * (float)(1000000.0f / F_CPU), 4);
}

float Platform::convertRawCoordinatesToDegrees(float RawDegrees) {
  float RawAsFloat = RawDegrees;
  int firstdigits = ((int)RawAsFloat) / 100; // Get the first digits by turning f into an integer, then doing an
integer divide by 100;
  float nexttwodigits = RawAsFloat - (float)(firstdigits * 100);
  float Converted = (float)(firstdigits + nexttwodigits / 60.0);
  return Converted;
}

void Platform::I2Cread(uint8_t Address, uint8_t Register, uint8_t Nbytes, uint8_t* Data)
{
  // Set register address
  Wire.beginTransmission(Address);
  Wire.write(Register);
  Wire.endTransmission();

  // Read Nbytes
  Wire.requestFrom(Address, Nbytes);
  uint8_t index = 0;
  while (Wire.available())
    Data[index++] = Wire.read();
}

void Platform::I2CwriteByte(uint8_t Address, uint8_t Register, uint8_t Data)
{
  // Set register address
  Wire.beginTransmission(Address);
  Wire.write(Register);
  Wire.write(Data);
  Wire.endTransmission();
}
```

1.4.4. Arduino.h

```
/*
  Arduino.h - Main include file for the Arduino SDK
  Copyright (c) 2005-2013 Arduino Team.  All right reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
*/

#ifndef Arduino_h
#define Arduino_h

#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <math.h>

#include <avr/pgmspace.h>
#include <avr/io.h>
#include <avr/interrupt.h>

#include "binary.h"

#ifdef __cplusplus
extern "C"{
#endif

void yield(void);

#define HIGH 0x1
#define LOW  0x0

#define INPUT 0x0
#define OUTPUT 0x1
#define INPUT_PULLUP 0x2

#define PI 3.1415926535897932384626433832795
#define HALF_PI 1.5707963267948966192313216916398
#define TWO_PI 6.283185307179586476925286766559
#define DEG_TO_RAD 0.017453292519943295769236907684886
#define RAD_TO_DEG 57.295779513082320876798154814105
#define EULER 2.718281828459045235360287471352

#define SERIAL  0x0
#define DISPLAY 0x1

#define LSBFIRST 0
#define MSBFIRST 1
```

```
#define CHANGE 1
#define FALLING 2
#define RISING 3

#if defined(__AVR_ATtiny24__) || defined(__AVR_ATtiny44__) || defined(__AVR_ATtiny84__)
  #define DEFAULT 0
  #define EXTERNAL 1
  #define INTERNAL1V1 2
  #define INTERNAL INTERNAL1V1
#elif defined(__AVR_ATtiny25__) || defined(__AVR_ATtiny45__) || defined(__AVR_ATtiny85__)
  #define DEFAULT 0
  #define EXTERNAL 4
  #define INTERNAL1V1 8
  #define INTERNAL INTERNAL1V1
  #define INTERNAL2V56 9
  #define INTERNAL2V56_EXTCAP 13
#else
#if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__) ||
defined(__AVR_ATmega1284__) || defined(__AVR_ATmega1284P__) || defined(__AVR_ATmega644__) ||
defined(__AVR_ATmega644A__) || defined(__AVR_ATmega644P__) ||
defined(__AVR_ATmega644PA__)
#define INTERNAL1V1 2
#define INTERNAL2V56 3
#else
#define INTERNAL 3
#endif
#define DEFAULT 1
#define EXTERNAL 0
#endif

// undefine stdlib's abs if encountered
#ifdef abs
#undef abs
#endif

#define min(a,b) ((a)<(b)?(a):(b))
#define max(a,b) ((a)>(b)?(a):(b))
#define abs(x) ((x)>0?(x):-(x))
#define constrain(amt,low,high) ((amt)<(low)?(low):((amt)>(high)?(high):(amt)))
#define round(x)     ((x)>=0?(long)((x)+0.5):(long)((x)-0.5))
#define radians(deg) ((deg)*DEG_TO_RAD)
#define degrees(rad) ((rad)*RAD_TO_DEG)
```

```
#define sq(x) ((x)*(x))

#define interrupts() sei()
#define noInterrupts() cli()

#define clockCyclesPerMicrosecond() ( F_CPU / 1000000L )
#define clockCyclesToMicroseconds(a) ( (a) / clockCyclesPerMicrosecond() )
#define microsecondsToClockCycles(a) ( (a) * clockCyclesPerMicrosecond() )

#define lowByte(w) ((uint8_t) ((w) & 0xff))
#define highByte(w) ((uint8_t) ((w) >> 8))

#define bitRead(value, bit) (((value) >> (bit)) & 0x01)
#define bitSet(value, bit) ((value) |= (1UL << (bit)))
#define bitClear(value, bit) ((value) &= ~(1UL << (bit)))
#define bitWrite(value, bit, bitvalue) (bitvalue ? bitSet(value, bit) : bitClear(value, bit))

// avr-libc defines _NOP() since 1.6.2
#ifndef _NOP
#define _NOP() do { __asm__ volatile ("nop"); } while (0)
#endif

typedef unsigned int word;

#define bit(b) (1UL << (b))

typedef bool boolean;
typedef uint8_t byte;

void init(void);
void initVariant(void);

int atexit(void (*func)()) __attribute__((weak));

void pinMode(uint8_t, uint8_t);
void digitalWrite(uint8_t, uint8_t);
int digitalRead(uint8_t);
int analogRead(uint8_t);
void analogReference(uint8_t mode);
void analogWrite(uint8_t, int);

unsigned long millis(void);
unsigned long micros(void);
```

```
void delay(unsigned long);
void delayMicroseconds(unsigned int us);
unsigned long pulseIn(uint8_t pin, uint8_t state, unsigned long timeout);
unsigned long pulseInLong(uint8_t pin, uint8_t state, unsigned long timeout);

void shiftOut(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder, uint8_t val);
uint8_t shiftIn(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder);

void attachInterrupt(uint8_t, void (*)(void), int mode);
void detachInterrupt(uint8_t);

void setup(void);
void loop(void);

// Get the bit location within the hardware port of the given virtual pin.
// This comes from the pins_*.c file for the active board configuration.

#define analogInPinToBit(P) (P)

// On the ATmega1280, the addresses of some of the port registers are
// greater than 255, so we can't store them in uint8_t's.
extern const uint16_t PROGMEM port_to_mode_PGM[];
extern const uint16_t PROGMEM port_to_input_PGM[];
extern const uint16_t PROGMEM port_to_output_PGM[];

extern const uint8_t PROGMEM digital_pin_to_port_PGM[];
// extern const uint8_t PROGMEM digital_pin_to_bit_PGM[];
extern const uint8_t PROGMEM digital_pin_to_bit_mask_PGM[];
extern const uint8_t PROGMEM digital_pin_to_timer_PGM[];

// Get the bit location within the hardware port of the given virtual pin.
// This comes from the pins_*.c file for the active board configuration.
//
// These perform slightly better as macros compared to inline functions
//
#define digitalPinToPort(P) ( pgm_read_byte( digital_pin_to_port_PGM + (P) ) )
#define digitalPinToBitMask(P) ( pgm_read_byte( digital_pin_to_bit_mask_PGM + (P) ) )
#define digitalPinToTimer(P) ( pgm_read_byte( digital_pin_to_timer_PGM + (P) ) )
#define analogInPinToBit(P) (P)
#define portOutputRegister(P) ( (volatile uint8_t *)( pgm_read_word( port_to_output_PGM + (P))) )
#define portInputRegister(P) ( (volatile uint8_t *)( pgm_read_word( port_to_input_PGM + (P))) )
#define portModeRegister(P) ( (volatile uint8_t *)( pgm_read_word( port_to_mode_PGM + (P))) )
```

```
#define NOT_A_PIN 0
#define NOT_A_PORT 0

#define NOT_AN_INTERRUPT -1

#ifdef ARDUINO_MAIN
#define PA 1
#define PB 2
#define PC 3
#define PD 4
#define PE 5
#define PF 6
#define PG 7
#define PH 8
#define PJ 10
#define PK 11
#define PL 12
#endif

#define NOT_ON_TIMER 0
#define TIMER0A 1
#define TIMER0B 2
#define TIMER1A 3
#define TIMER1B 4
#define TIMER1C 5
#define TIMER2  6
#define TIMER2A 7
#define TIMER2B 8

#define TIMER3A 9
#define TIMER3B 10
#define TIMER3C 11
#define TIMER4A 12
#define TIMER4B 13
#define TIMER4C 14
#define TIMER4D 15
#define TIMER5A 16
#define TIMER5B 17
#define TIMER5C 18

#ifdef __cplusplus
} // extern "C"
#endif
```

```
#ifdef __cplusplus
#include "WCharacter.h"
#include "WString.h"
#include "HardwareSerial.h"
#include "USBAPI.h"
#if defined(HAVE_HWSERIAL0) && defined(HAVE_CDCSERIAL)
#error "Targets with both UART0 and CDC serial not supported"
#endif

uint16_t makeWord(uint16_t w);
uint16_t makeWord(byte h, byte l);

#define word(...) makeWord(__VA_ARGS__)

unsigned long pulseIn(uint8_t pin, uint8_t state, unsigned long timeout = 1000000L);
unsigned long pulseInLong(uint8_t pin, uint8_t state, unsigned long timeout = 1000000L);

void tone(uint8_t _pin, unsigned int frequency, unsigned long duration = 0);
void noTone(uint8_t _pin);

// WMath prototypes
long random(long);
long random(long, long);
void randomSeed(unsigned long);
long map(long, long, long, long, long);

#endif

#include "pins_arduino.h"

#endif
```

1.4.5. Display.cpp

```
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Max72xxPanel.h>

Max72xxPanel matrix = Max72xxPanel(5, 1, 1);
int wait = 100; // In milliseconds
int spacer = 1;
int width = 5 + spacer; // The font width is 5 pixels
```

```
String stringUART = ""; // Переменная сбора принятых командных символов в строку
bool startedUART; // переменная начала приема командных данных по uart
uint8_t indexUART = 0; // Индекс принятого аргумента командного режима

byte mask[8] = {
  0b00000000,
  0b00000000,
  0b00000000,
  0b00000000,
  0b00000000,
  0b00000000,
  0b00000000,
  0b00000000
};

String receivedTicker = "";

void setup() {
  Serial.begin(115200);
  //ticker("");
  //pixelsDraw();
  matrix.fillScreen(LOW);
  matrix.write();
}

void loop() {
  if (Serial.available() > 0) {
    char incomingByte = Serial.read();
    if (incomingByte != ',' && incomingByte != ';') stringUART += incomingByte;
    else {
      switch (indexUART) {
        case 0:
          receivedTicker = stringUART;
          receivedTicker.replace(":", "");
          break;
        case 1:
          mask[0] = (byte)stringUART.toInt();
          break;
        case 2:
          mask[1] = (byte)stringUART.toInt();
          break;
        case 3:
```

```
        mask[2] = (byte)stringUART.toInt();
        break;
      case 4:
        mask[3] = (byte)stringUART.toInt();
        break;
      case 5:
        mask[4] = (byte)stringUART.toInt();
        break;
      case 6:
        mask[5] = (byte)stringUART.toInt();
        break;
      case 7:
        mask[6] = (byte)stringUART.toInt();
        break;
      case 8:
        mask[7] = (byte)stringUART.toInt();
        break;
    }
    stringUART = "";                 // очищаем строку
    indexUART++;                      // переходим к парсингу следующего элемента массива
   }
   if (incomingByte == '*') {                  // если это *
    startedUART = true;          // поднимаем флаг, что можно парсить
    indexUART = 0;                    // сбрасываем индекс
    stringUART = "";                  // очищаем строку
   }
   if (incomingByte == ';') {  // если таки приняли ; - конец парсинга
    startedUART = false;          // сброс
    if(receivedTicker.length() > 0) ticker(receivedTicker);
    else pixelsDraw();
   }
  }
}

void pixelsDraw() {
  for (int y = 0; y < 8; y++ ) {       // Передача массива
   for (int x = 0; x < 8; x++ ) {
    matrix.drawPixel(x, y, mask[y] & (1 << x));
   }
  }
  matrix.write();
}
```

```
void ticker(String tape) {
  for ( int i = 0 ; i < width * tape.length() + matrix.width() - spacer; i++ )
  {
    matrix.fillScreen(LOW);

    int letter = i / width;                  // номер символа выводимого на матрицу

    int x = (matrix.width() - 1) - i % width;
    int y = (matrix.height() - 8) / 2;       // отцентрировать текст по вертикали

    while ( x + width - spacer >= 0 && letter >= 0 ) {
      if ( letter < tape.length() ) {
        matrix.drawChar(x, y, tape[letter], HIGH, LOW, 1);
      }
      letter--;
      x -= width;
    }
    matrix.write();                          // выведим значения на матрицу
    delay(wait);
  }
  receivedTicker = "";
}
```