Inteligentny kask rowerowy

Idea

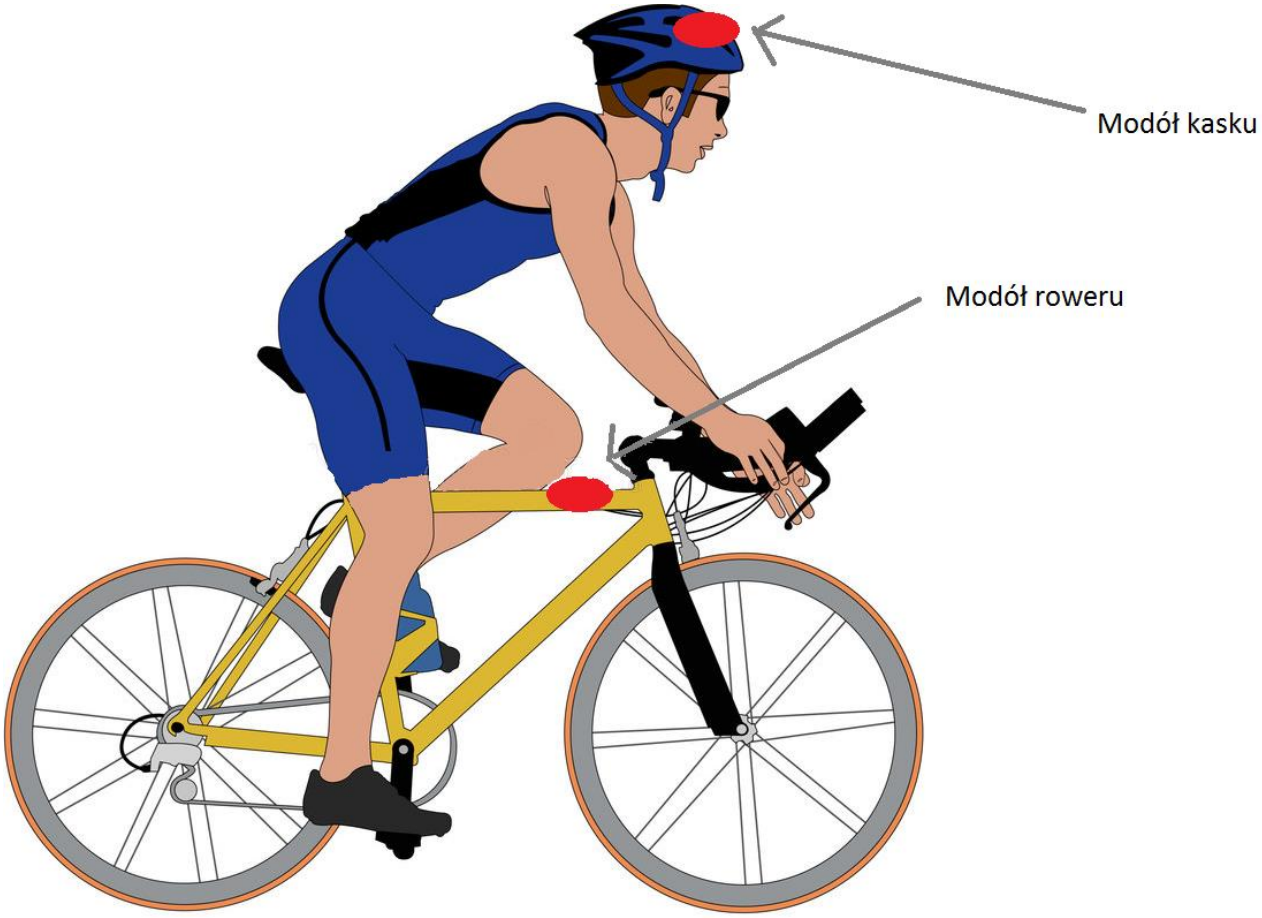
W dzisiejszych czasach coraz większą uwagę poświęca się ekologii oraz zdrowemu trybowi życia, coraz więcej ludzi sięga po rower jako środek lokomocji. Jednak wiele miast nie jest przygotowanych na przyjęcia wzrastającej liczby rowerzystów. W związku z tym są oni zmuszeni do poruszania się po ruchliwych ulicach, po których dziennie przejeżdżają setki samochodów. Sami także jesteśmy rowerzystami i chcielibyśmy na drogach czuć się bezpiecznie. Dlatego opracowaliśmy projekt i prototyp kasku ze światłami i kierunkowskazami.

Elektronika i oprogramowanie

Elektronika urządzenia składa się z dwóch części modułu na kasku (odbiornika) i modułu zamontowanego do ramy roweru (nadajnika).

a)**modułu kasku** jest sterowany mikroprocesorem atmega328 na płytce Arduino Nano z modułem bluetooth (HC-05) i kompasem elektronicznym (LSM303) oraz sześcioma paskami LED RGB ze sterownikiem WS2812. Pomimo niskiej wydajności komputera, możliwa jest wystarczająca częsta analiza danych wejściowych i zapalanie odpowiednich diod – dzięki wykorzystaniu programu napisanego w języku C.

b)**modułu roweru** bazuje również na arduino nano z modułem bluetooth (HC-05) i kompasem elektronicznym (LSM303) oraz dwoma spustami wykonanymi z krańcówek służącymi do włączania kierunkowskazów.



Źródło zdjęcia: Dreamstime.com

Opis urządzenia

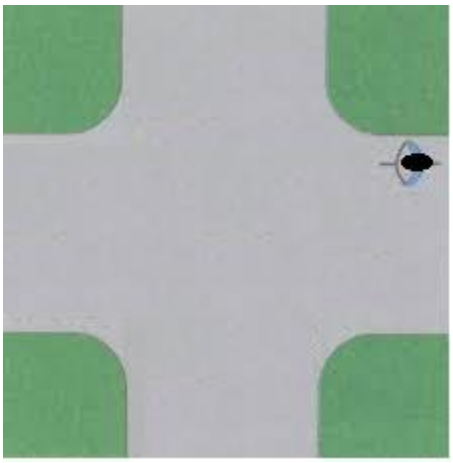
Urządzenie jest zbudowane z myślą o rowerzystach. Podczas konstruowania mieliśmy w głowie jedną myśl, jeden cel – poprawić bezpieczeństwo rowerzystów na drogach oraz zmniejszenie ilości wypadków i kolizji drogowych z ich udziałem. Podczas budowy prototypu wykorzystaliśmy typowy kask rowerowy zakupiony w jednym

z supermarketów, mikroprocesor atmega328, kompasy elektroniczny lsm303 oraz diody LED RGB ze sterownikami WS2812.

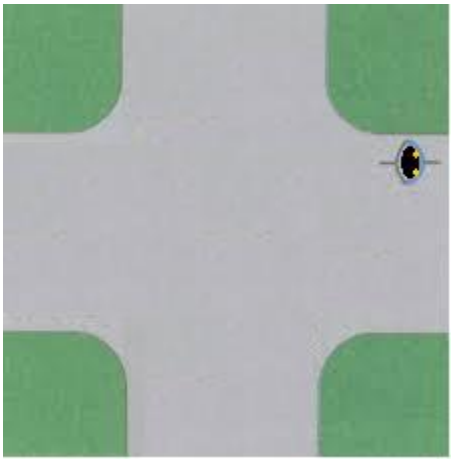
Zasada działania :założeniem projekt jest kask który po skręceniu głowy w lewo lub w prawo włącza automatycznie odpowiedni kierunkowskaz i wyłącza go po wykonaniu zkrętu przez rowerzystę. Jeżeli chodzi o szczegóły teshniczne to procesor zamontowany na ramie roweru odczytuje pozycję kompasu przez interfejs I2C, przetwarza odczytane dane na pozycję w stopniach (0-360) odczytane dane wysyła do kasku za pomocą bluetootha, który porównuje je z odczytami kompasu zamieszczonym na kasku i określa różnice w stopniach między aktualną pozycją kasku i ramy roweru jeżeli jest ona większa niż 60° lub mniejsza niż -60° wtedy włącza kierunkowskaz odpowiednio lewy lub prawy, dalej kierunkowskaz jest wyłączony po tym jak rower dokona skrętu (wartość odczytana z kompasu na ramie roweru osiągnie wartość powiększoną (lewo) o 60° lub pomniejszoną (prawo) o 60° w stosunku do wartości odczytu z przed skrętu na tej podstawie wie że rowerzysta skręcił. Częstotliwość pracy układu to około 10 Hz w tym wliczony czas na odczytanie danych z obydwu kompasów, przesłanie danych do kasku oraz ich porównanie więcej szczegółów znajduje się w komentarzach dodanych do bibliotek i programu.

Rysunek poglądowy skrętu w prawo

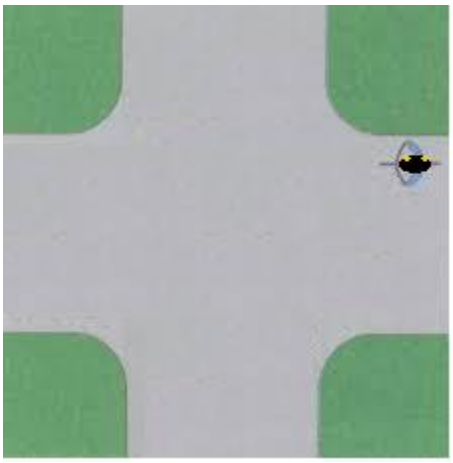
Rowerzysta podjeżdża do skrzyrzowania



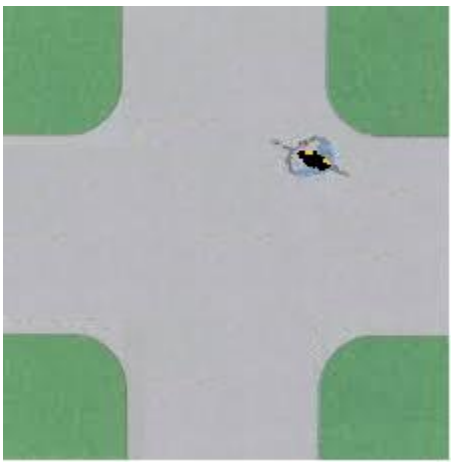
Rowerzysta skręca głowę w prawo kierunkowskaz włącza się



Rowerzysta upewnia się czy nic nie nadjeżdża



Rowerzysta wykonuje manewr skrętu



Po wykonabym manewrze kierunkowskaz wyłącza się

Założenia konstrukcyjne

Projektując i konstruując kask kierowały nami następujące założenia:

- ✓ niskie koszty produkcji;
- ✓ prosta obsługa;
- ✓ precyzja działania;
- ✓ wykorzystanie układów programowalnych używanych na lekcjach oraz tworzenie znanym z zajęć środowisku programistycznym jakim jest Eclipse Mars;
- ✓ zwiększenie bezpieczeństwa rowerzystów.

Wnioski

Po wykonaniu prototypu urządzenia oraz przeanalizowaniu jego działania zaskoczyła nas stosunkowo duża precyzja odczytów kompasu oraz płynność działania inteligentnego kasku.

Dodanie kierunkowskazów aktywowanych poprzez spusty na kierownicy było trafionym pomysłem. Rozwiązanie to, jest wygodne, zarówno dla rowerzystów jak i kierowców.

Kosztorys

1. 2x Arduino Nano 29,90zł sztuka
2. elektronika:

✓ 2x kompas elektroniczny LSM303, 30,90zł sztuka

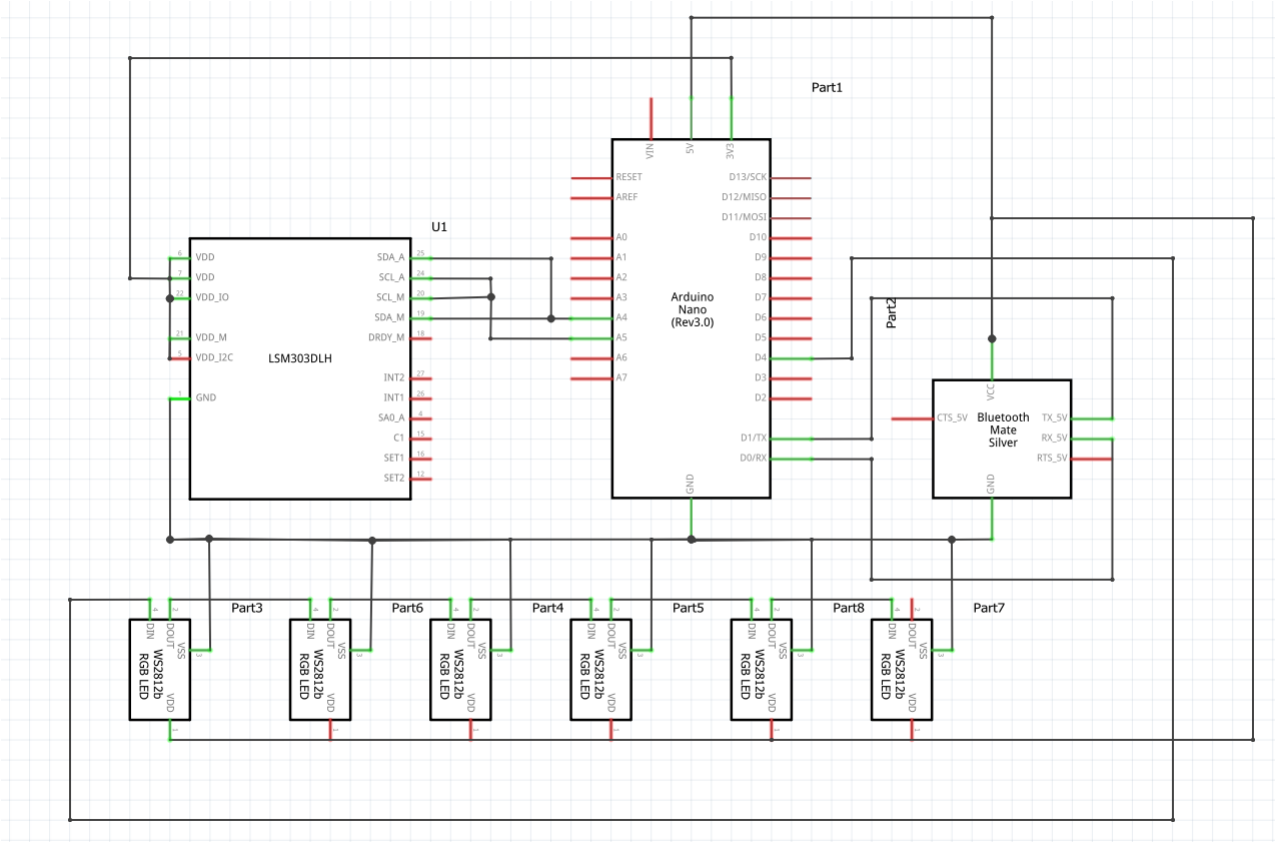
✓ 2x moduł bluetootha HC-05, 35,90 zł sztuka

✓ 6x paski LED RGB ze sterownikiem WS2812, 17,90 zł sztuka

3. kask rowerowy 20 zł

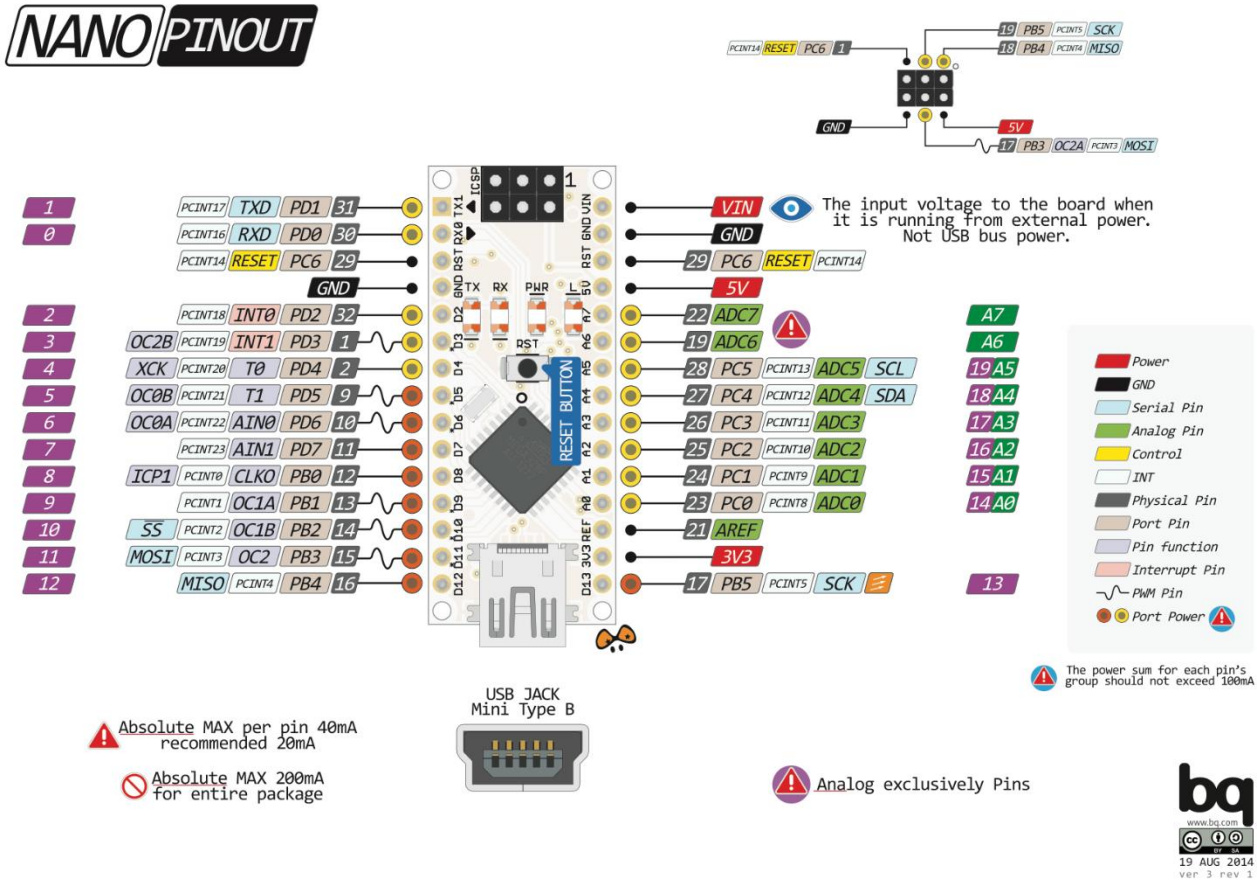
koszt całości wynosi 320,8 zł dla cen jednostkowych produktów.

Schemat połączenia modułu kasku i ramy roweru

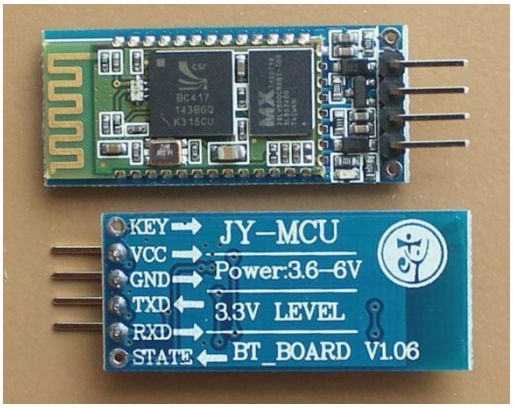


Dokumentacja techniczna

Wyprowadzenia Arduino Nano:



Modułu bluetooth HC-05



Kompasu lsm303

Oraz linki do dokumentacji

Kompas lsm303

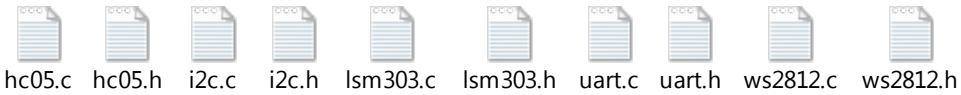
http://www.st.com/content/ccc/resource/technical/document/datasheet/56/ec/ac/de/28/21/4d/48/DM00027543.pdf/files/DM00027543.pdf/jcr:content/translations/en.DM00027543.pdf

poradnik dla kompasu hmc5883l

http://www.jarzebski.pl/arduino/czujniki-i-sensory/3-osiowy-magnetometr-hmc5883l.html

Poniżej znajdują się biblioteki oraz główne pętle programów

Bibilioteki:

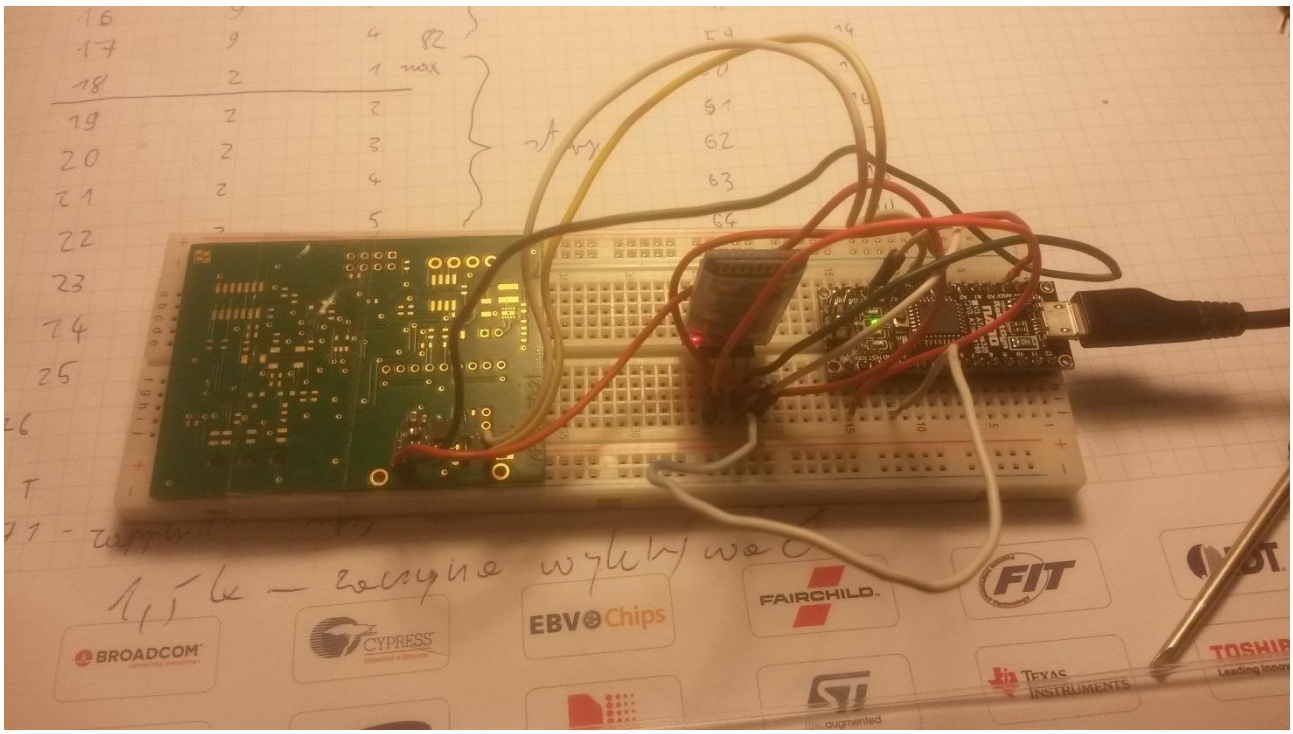


Pętla główna nadajnika (ramy)

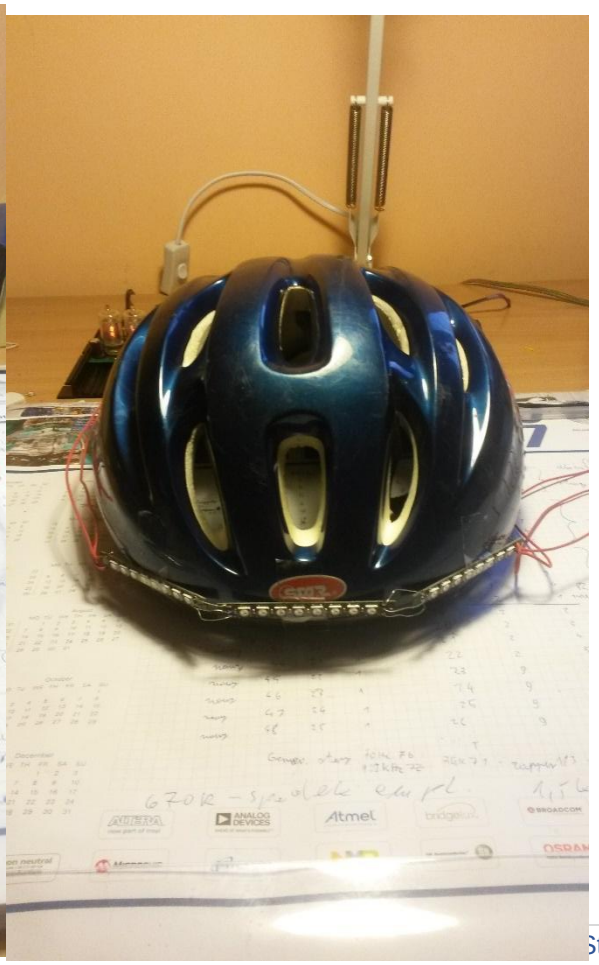
Pętla główna odbiornika (kasku)

Wszystkie programy powyżej zostały napisane samodzielnie ze wspieraniem się notą katalogową lsm303 i poradnikiem dla hmc5883l z którego pozyskałem wzór na odczytanie kąta kompasu

Pierwszą wersję prototyp zdecydowłem się wykonać na płytce stykowej



A oto gotowy prototyp kasku



Treść bibliotek i programów

Biblioteka I2C

i2c.h

```
#ifndef I2C_H_
```

```
#define I2C_H_

#include <avr/io.h>
#include <util/delay.h>


#if defined(_AVR_ATtiny44A_H_)
#define i2c_ddr DDRA
#define i2c_pin PINA
#define i2c_port PORTA
#define scl_port PA4
#define sda_port PA5
#endif

#if defined(__AVR_ATmega328P__)
#define i2c_ddr DDRB
#define i2c_pin PINB
#define i2c_port PORTB
#define scl_port PB0
#define sda_port PB1
#endif

#if defined(__AVR_ATmega32__)
#define i2c_ddr DDRC
#define i2c_pin PINC
#define i2c_port PORTC
#define scl_port PC0
#define sda_port PC1
#endif

#if defined(__AVR_ATmega16__)
#define i2c_ddr DDRC  //bPn8aKDPGe4HkJ5p
#define i2c_pin PINC
#define i2c_port PORTC
#define scl_port PC0
#define sda_port PC1
#endif

#define scl_out i2c_ddr |= (1<<scl_port)
#define scl_one i2c_port |= (1<<scl_port)
#define scl_zero i2c_port &= ~(1<<scl_port)
#define scl_in i2c_ddr &= ~(1<<scl_port)
```

```
#define get_scl (i2c_pin & (1<<scl_port))
#define sda_out i2c_ddr |= (1<<sda_port)
#define sda_one i2c_port |= (1<<sda_port)
#define sda_zero i2c_port &= ~(1<<sda_port)
#define sda_in i2c_ddr &= ~(1<<sda_port)
#define get_sda !(i2c_pin & (1<<sda_port))

#define delay1 _delay_us(2)
#define delay2 _delay_us(4)



        void i2c_init(void);
        void i2c_start(void);
        void i2c_stop(void);
        void i2c_requestFrom(uint8_t addr);
        void i2c_sendTo(uint8_t addr);
        uint8_t i2c_send_byte(uint8_t data);
        void i2c_send(uint8_t slaveAddr,uint8_t data,uint8_t stop);
        uint8_t i2c_read(uint8_t slaveAddr,uint8_t stop);
        uint8_t i2c_read_byte(uint8_t stop);

#endif /* I2C_H_ */
```

i2c.c

```
#include "i2c.h"
#include <avr/io.h>
#include <util/delay.h>

void i2c_init(void){
        scl_out;
        sda_out;
        scl_one;
        sda_one;
}
void i2c_start(void){
        scl_out;
        sda_out;
        delay2;
        scl_one;
        sda_one;
        delay1;
```

```
	sda_zero;
	delay1;
	scl_zero;
}
void i2c_stop(void){
	delay1;
	sda_out;
	sda_zero;
	delay1;
	scl_one;
	delay1;
	sda_one;
}
void i2c_requestFrom(uint8_t addr){
	i2c_send_byte((addr<<1)|1);
}
void i2c_sendTo(uint8_t addr){
	i2c_send_byte((addr<<1)|0);
}
uint8_t i2c_send_byte(uint8_t data){
	uint8_t i;
	sda_out;
	delay1;
	for(i=0;i<8;i++){
			if((data>>(7-i)) & 1)sda_one;
			else sda_zero;
			delay1;
			scl_one;
			delay2;
			scl_zero;
			delay1;
	}
	sda_in;
	delay1;
	scl_one;
	if(get_sda & 1)i=1;
	else i=0;
	delay2;
	scl_zero;
	if(i)return 1;
	else return 0;
}
```

```
void i2c_send(uint8_t slaveAddr,uint8_t data,uint8_t stop){
	//		cli();
	//uint8_t i;
	i2c_start();
	i2c_send_byte((slaveAddr<<1)|0);
	i2c_send_byte(data);
	//if(!i)led2_off;
	if(stop)i2c_stop();
//		sei();
}
uint8_t i2c_read(uint8_t slaveAddr,uint8_t stop){
	i2c_start();
	uint8_t i,c,val=0;
	i = i2c_send_byte((slaveAddr<<1)|1);
	sda_in;
	sda_one;
	for(i=0;i<8;i++){
		delay2;
		scl_one;
		delay1;
		c = get_sda;
		if(c==1){
			//led2_off;
			val |= (1<<(7-i));
		}
		delay1;
		scl_zero;
	}
	delay1;
	sda_out;
	sda_zero;
	delay1;
	scl_one;
	delay2;
	scl_zero;
	if(stop)i2c_stop();
	val = ~val;
	return val;
}
uint8_t i2c_read_byte(uint8_t stop){
	uint8_t i,c,val=0;
	sda_in;
```

```
        sda_one;
        for(i=0;i<8;i++){
                delay2;
                scl_one;
                delay1;
                c = get_sda;
                if(c==1){
                        //led2_off;
                        val |= (1<<(7-i));
                }
                delay1;
                scl_zero;
        }
        delay1;
        sda_out;
        if(stop)sda_zero;
        else sda_one;
        delay1;
        scl_one;
        delay2;
        scl_zero;
        delay1;
        if(stop)sda_zero;
        delay1;
        scl_one;
        delay2;
        scl_zero;
        if(stop)i2c_stop();
        val = ~val;
        return val;
}
```

biblioteka UART

uart.h

```
 #ifndef UART_UART_H_
#define UART_UART_H_
#include <avr/io.h>
#include <util/delay.h>
#define F_CPU 16000000UL
#define BAUDRATE 9600
#define BAUD_PRESCALLER (((F_CPU / (BAUDRATE * 16UL))) - 1)
```

```
void USART_init(void);
void USART_send_nr(int nr);
unsigned char USART_receive(void);
void USART_send( unsigned char data);
void USART_putstring(char* StringPtr);

#endif /* UART_UART_H_ */
```

uart.c

```
#include "uart.h"
#include <avr/io.h>
#include <util/delay.h>

void uart_send_nr(int nr){
        char str[16];
        itoa(nr, str, 10);
        uart_putstring(str);
        uart_send('\n');
}
void uart_init(void){
UBRR0H = (uint8_t)(BAUD_PRESCALLER>>8);
 UBRR0L = (uint8_t)(BAUD_PRESCALLER);  // 9600 bps
 UCSR0B = (1<<RXEN0)|(1<<TXEN0);  //włączenie linii rx i tx
 UCSR0C = (3<<UCSZ00);  //tryb asynchroniczny słowo 8 bitowe
}

unsigned char uart_receive(void){

 while(!(UCSR0A & (1<<RXC0)));  //nowe dane odebrane
 return UDR0;

}

void uart_send( unsigned char data){

 while(!(UCSR0A & (1<<UDRE0)));  //czeka bufor odbieranych danych będzie pusty
 UDR0 = data;

}

void uart_putstring(char* StringPtr){

while(*StringPtr != 0){
```

```
 uart_send(*StringPtr);
 StringPtr++;
}
}
```

biblioteka WS2812

ws2812.h

```
z #ifndef WS2812_WS2812_H_
#define WS2812_WS2812_H_
#include <avr/io.h>
#include <util/delay.h>

/* ustawienia portu */
#define ws2812_port PORTD
#define ws2812_ddr DDRD
#define ws2812_pin PD4
#define ws2812_port_init ws2812_ddr |= (1<<ws2812_pin)
#define set_1 ws2812_port |= (1<<ws2812_pin)
#define set_0 ws2812_port &= ~(1<<ws2812_pin)

// częotliwośkwarcu procesora to 16MHz co daj nam 1 cykl zegarowy o czasie 62 ns
// T0H to 6 cykli
#define T0H asm volatile ("nop");asm volatile ("nop");asm volatile ("nop");asm volatile ("nop");asm volatile
("nop");asm volatile ("nop")

// T1H to 13 cykli
#define T1H asm volatile ("nop");asm volatile ("nop");asm volatile ("nop");asm volatile ("nop");asm volatile
("nop");asm volatile ("nop");asm volatile ("nop");asm volatile ("nop");asm volatile ("nop");asm volatile
("nop");asm volatile ("nop");asm volatile ("nop");asm volatile ("nop")

// T0L to 14 cykli
#define T0L asm volatile ("nop");asm volatile ("nop");asm volatile ("nop");asm volatile ("nop");asm volatile
("nop");asm volatile ("nop");asm volatile ("nop");asm volatile ("nop");asm volatile ("nop");asm volatile
("nop");asm volatile ("nop");asm volatile ("nop");asm volatile ("nop");asm volatile ("nop")

// T1L to 7 cykli
#define T1L asm volatile ("nop");asm volatile ("nop");asm volatile ("nop");asm volatile ("nop");asm volatile
("nop");asm volatile ("nop");asm volatile ("nop")

#define send_0 set_1;T0H;set_0;T0L;set_1  // wysyła logiczne "0" do szeregu diod
#define send_1 set_1;T1H;set_0;T1L;set_1  // wysyła logiczne "1" do szeregu diod
```

```
#define reset set_0;_delay_us(55);set_1

#define quantity 48 // ilość diod
uint8_t red[quantity],green[quantity],blue[quantity]; // tablice globalne przechowują dane które są wysłane do diod

void ws2812_init(void);  // ustawinie portu jako wyjście i wyzerowanie tablic
void ws2812_set_single(uint8_t poz,uint8_t r,uint8_t g,uint8_t b);
void ws2812_send(void);

#endif /* WS2812_WS2812_H_ */
```

ws2812.c

```
#include "ws2812.h"
#include <avr/io.h>
#include <util/delay.h>

void ws2812_init(void){
	ws2812_port_init;
	uint8_t i;
	for(i=0;i<quantity;i++){
		red[i]=0;green[i]=0;blue[i]=0;
	}
}
void ws2812_set_single(uint8_t poz,uint8_t r,uint8_t g,uint8_t b){
	red[poz]=r;green[poz]=g;blue[poz]=b;
}
void ws2812_send(void){
	uint8_t i,a;
	for(i=0;i<quantity;i++){
		for(a=0;a<7;a++){
			if(((green[i]>>a)&1)==1){
				send_1;
			}else send_0;
		}
		for(a=0;a<7;a++){
			if(((red[i]>>a)&1)==1){
				send_1;
			}else send_0;
		}
		for(a=0;a<7;a++){
			if(((blue[i]>>a)&1)==1){
```

```
                                send_1;
                        }else send_0;
                }
        }
        reset;
}
```

biblioteka lsm303

lsm303.h

```
#ifndef LSM303_LSM303_H_
#define LSM303_LSM303_H_
#include <avr/io.h>
#include <util/delay.h>
#include "../i2c/i2c.h"

#define OUT_X_L_A 0x28
#define OUT_X_H_A 0x29
#define OUT_Y_L_A 0x2A
#define OUT_Y_H_A 0x2B
#define OUT_Z_L_A 0x2C
#define OUT_Z_H_A 0x2D

#define MagWrite_Addr 0x3C
#define MagRead_Addr 0x3D
#define AccWrite_Addr 0x32
#define AccRead_Addr 0x33


#define CTRL_REG1_A 0x20
#define CTRL_REG2_A 0x21
#define CTRL_REG3_A 0x22
#define CTRL_REG4_A 0x23
#define CTRL_REG5_A 0x24
#define CTRL_REG6_A 0x25
#define CRA_REG_M 0x00
#define CRB_REG_M 0x01
#define MR_REG_M 0x02

float lsm303_heading(void);
uint16_t lsm303_read_y(void);
uint16_t lsm303_read_x(void);
```

```
uint16_t lsm303_read_z(void);
void lsm303_writeAccReg(uint8_t addr,uint8_t data);
void lsm303_writeMagReg(uint8_t addr,uint8_t data);
uint8_t lsm303_readAccReg(uint8_t addr);
uint8_t lsm303_readMagReg(uint8_t addr);
void lsm303_init(void);

#endif /* LSM303_LSM303_H_ */
```

lsm303.c

```
#include "lsm303.h"
#include <avr/io.h>
#include <util/delay.h>
#include "../i2c/i2c.h"
#include "math.h"

float lsm303_heading(void){  // aktualny stan kompasu
        uint16_t x,y;
        x = lsm303_read_x();
        y = lsm303_read_y();
        float PI = 3.14;
        // Obliczenie kierunku (rad)
        float heading = atan2(y,x);
          // Ustawienie kata deklinacji dla Lublina 6'2E (positive)
          // Formula: (deg + (min / 60.0)) / (180 / M_PI);
          float declinationAngle = (6.0 + (2.0 / 60.0)) / (180 / M_PI);
          heading += declinationAngle;
          // Korekta katow
          if (heading < 0)heading += 2 * PI;

          if (heading > 2 * PI)heading -= 2 * PI;
          // Zamiana radianow na stopnie
          float headingDegrees = heading * 180/M_PI;
        return headingDegrees;
}
uint16_t lsm303_read_x(void){  // wektor x
        uint8_t x_l,x_h;
        uint16_t out_x;
        x_l = lsm303_readMagReg(OUT_X_L_A);
        x_h = lsm303_readMagReg(OUT_X_H_A);
        out_x = (x_l + (x_h<<8));
        return out_x;
}
```

```
uint16_t lsm303_read_y(void){  // wektor y
        uint8_t y_l,y_h;
        uint16_t out_y;
        y_l = lsm303_readMagReg(OUT_Y_L_A);
        y_h = lsm303_readMagReg(OUT_Y_H_A);
        out_y = (y_l + (y_h<<8));
        return out_y;
}
uint16_t lsm303_read_z(void){  // wektor z
        uint8_t z_l,z_h;
        uint16_t out_z;
        z_l = lsm303_readMagReg(OUT_Z_L_A);
        z_h = lsm303_readMagReg(OUT_Z_H_A);
        out_z = (z_l + z_h)/2;
        return out_z;
}
void lsm303_writeAccReg(uint8_t addr,uint8_t data){ // zapis do rejestru akcelerometru
        i2c_start();
        i2c_send_byte(AccWrite_Addr);
        i2c_send_byte(addr);
        i2c_send_byte(data);
        i2c_stop();
}
uint8_t lsm303_readAccReg(uint8_t addr){  // odczyt z rejestru akcelerometru
        i2c_start();
        i2c_send_byte(AccWrite_Addr);
        i2c_send_byte(addr);
        i2c_start();
        i2c_requestFrom(AccRead_Addr);
        uint8_t ret=i2c_read_byte(0);
        i2c_stop();
        return ret;
}
void lsm303_writeMagReg(uint8_t addr,uint8_t data){ // zapis do rejestru magnetometru
        i2c_start();
        i2c_send_byte(MagWrite_Addr);
        i2c_send_byte(addr);
        i2c_send_byte(data);
        i2c_stop();
}
uint8_t lsm303_readMagReg(uint8_t addr){  // odczyt z rejestru magnetometru
        i2c_start();
```

```
	i2c_send_byte(MagWrite_Addr);
	i2c_send_byte(addr);
	i2c_start();
	i2c_requestFrom(MagRead_Addr);
	uint8_t ret=i2c_read_byte(0);
	i2c_stop();
	return ret;
}
void lsm303_init(void){  // rozpoczcie odczytu ustawienie trybu pracy
	i2c_init();
	lsm303_writeAccReg(CTRL_REG4_A, 0x08);
	lsm303_writeAccReg(CTRL_REG1_A, 0x47);
	lsm303_writeMagReg(CRA_REG_M, 0x0C);
	lsm303_writeMagReg(CRB_REG_M, 0x20);
	lsm303_writeMagReg(MR_REG_M, 0x00);
}
```

biblioteka HC05

hc05.h

```
#ifndef HC05_HC05_H_
#define HC05_HC05_H_
#include <avr/io.h>
#include <util/delay.h>
#include "../uart/uart.h"

uint8_t state=0;
uint16_t counter=0,data[3];

void HC05_begin();
uint16_t HC05_recive();
void HC05_send(uint16_t value);

#endif /* HC05_HC05_H_ */
```

hc05.c

```
#include "hc05.h"
#include <avr/io.h>
#include <util/delay.h>
#include "../uart/uart.h"
```

```
void HC05_begin(){
	uart_init();
}
uint16_t HC05_recive(){
	uint8_t done=0;
	while(done==0){
		int recived = uart_receive();
		while(recived==-1)recived = uart_receive();
		if(state){
			data[counter]=recived;
			counter++;
			if(counter==3){
				counter=0;
				state=0;
				done=1;
			}
		}
		if(recived==3){
			state=1;
		}
	}
	return (data[2]*100)+(data[1]*10)+data[0];
}
void HC05_send(uint16_t value){
	int current;
	uint8_t i;
	uart_send_nr(3);
	for(i=0;i<3;i++){
		_delay_ms(6);
		current=value/(pow(10,i));
		uart_send_nr(current%10);
	}
	_delay_ms(16);
}
```

Pętla programu kasku

```
#include <avr/io.h>
#include <util/delay.h>
#include "hc05/hc05.h"
#include "i2c/i2c.h"
```

```
#include "lsm303/lsm303.h"
#include "uart/uart.h"

int val_difference,val_difference_from_recived,recv,tim_switch;
uint8_t state;
int main(){
        ws2812_init();
        lsm303_init();
        HC05_begin();
        val_difference = val_current();
         recv=HC05_recive();      //odczyt z ramy roweru
         while(recv<0||recv>360){
           recv=HC05_recive();   //sprawdzenie poprawnoci danych odczytanych
         }
         val_difference_from_recived = recv; //zmienna od kttórej liczona jest różnica stopni tak aby na
początku 2 kompasy wskazywaly 0stopni
         for(int i=0;i<23;i++){
           setWs(100,100,100,i);
         }
         ws2812_send();
         noInterrupts();          // wylanczenie przerwania  TCCR1A = 0;
         TCCR1B = 0;
         TCNT1  = 0;

  OCR1A   = 31250;           // rejestr ustalenia częstotliwosć pracy czyli 2Hz
  TCCR1B |= (1 << WGM12);   // CTC mode
  TCCR1B |= (1 << CS12);    // 256 prescaler
  TIMSK | = (1 << OCIE1A);  // włączenie przerwania lokalnego
         interrupts();
         tim_switch=0;
        }
        ISR(TIMER1_COMPA_vect)          // czynnosci przy przerwaniu
        {
         if(tim_switch==1){  // kierunkowskaz lewy miganie
         if(state){
         for(int i=0;i<48;i++){
          if(i<8)setWs(150,100,0,i);
          if(i<32&&i>=24)setWs(150,100,0,i);
          else setWs(100,100,100,i);
         }
         ws2812_send();
         }
```

```
  else{
    for(int i=0;i<48;i++){
     setWs(100,100,100,i);
  }
  ws2812_send();
  }
  state = !state;
}else if(tim_switch==2){  // kierunkowskaz prawy miganie
  if(state){
  for(int i=0;i<48;i++){
    if(i>16&&i<24)setWs(150,100,0,i);
    if(i>40)setWs(150,100,0,i);
    else setWs(100,100,100,i);
  }
  ws2812_send();
  }
  else{
    for(int i=0;i<48;i++){
     setWs(100,100,100,i);
  }
  ws2812_send();
  }
  state = !state;
}
 else{                // brak kierunkowskazu
  for(int i=0;i<48;i++){
     setWs(100,100,100,i);
  }
  ws2812_send();
}
}
 void loop()
{
  int val_c = val_current();    // odczyt aktualnej wartosci z kasku
  int val_at_0 = val_at_zero(val_c);  // przyrównanie do 0
  recv=HC05_recive();                          //odczyt z ramy roweru
  while(recv<0||recv>360){
     recv=HC05_recive();
  }
  int recv_at_0 = val_at_zero_recived(recv); // przyrównanie do 0
  int dif = difference(val_at_0,recv_at_0);  // wyliczenie różnicy
  if(dif>60){  //jesli wynosi +60 kierunkowskaz prawy
```

```
    tim_switch=2;
    if((recv_at_0+60)<360)turn_Right(recv_at_0+60); // ustawienie wartosci przy której program się
resetuje
    else turn_Right((recv_at_0+60)-360);
   }
   if(dif<(-60)){ //jesli różnica wynosi -60 kierunkowskaz prawy
    tim_switch=1;
    if((recv_at_0-60)>0)turn_Left(recv_at_0-60); // ustawienie wartosci przy której program się
resetuje
    else turn_Left(360-((recv_at_0-60)*(-1)));
   }
   _delay_ms(100);
  }
  void turn_Left(int reset_val){
   tim_switch=1;
   digitalWrite(2,1);
   int recv_at_0;
   back:
   recv=HC05_recive();
   while(recv<0||recv>360){
     recv=HC05_recive();
   }
   recv_at_0 = val_at_zero_recived(recv);
   int dif = difference(recv_at_0,reset_val);
   if(dif>0)goto back;
   tim_switch=0;
   asm volatile("jmp 0");
  }
  void turn_Right(int reset_val){
   int recv_at_0;
   back:
   recv=HC05_recive();
   while(recv<0||recv>360){
     recv=HC05_recive();
   }
   recv_at_0 = val_at_zero_recived(recv);
   int dif = difference(reset_val,recv_at_0);
   if(dif>0)goto back;
   tim_switch=0;
   asm volatile("jmp 0");
  }
```

```
int difference(int kask,int kierownica){
  int ret_val;
  if(kask>kierownica&&(kask-kierownica)>0&&(kask-kierownica)<180)ret_val = kask-kierownica;
  if(kask<kierownica&&(kierownica-kask)>0&&(kierownica-kask)<180)ret_val = (kierownica-kask)*(-
1);
  if(kask>kierownica&&(kask-kierownica)>0&&(kask-kierownica)>180)ret_val = (((kask-360)*(-
1))+kierownica)*(-1);
  if(kask<kierownica&&(kierownica-kask)>0&&(kierownica-kask)>180)ret_val = (((kierownica-360)*(-
1))+kask);
  if(kask==kierownica)ret_val=0;
  return ret_val;
}
int val_at_zero_recived(int current){
if(current >= val_difference_from_recived)return (current-val_difference_from_recived);
if(current < val_difference_from_recived)return (360+(current-val_difference_from_recived));
}
int val_at_zero(int current){
if(current >= val_difference){
        return (current-val_difference);
}
if(current < val_difference){
        return (360+(current-val_difference));
}
}
int val_current(){
  float heading = lsm303_heading();
return heading;
}
void setWs(int r,int g,int b,int led_nr){
red[led_nr] = r;
green[led_nr] = g;
blue[led_nr] = b;
}
```

Pętla programu ramy roweru

```
#include <avr/io.h>
#include <util/delay.h>
#include "hc05/hc05.h"
#include "i2c/i2c.h"
#include "lsm303/lsm303.h"
#include "uart/uart.h"
```

```
int main(){
	lsm303_init();
	HC05_begin();
	while(1){
		uint16_t heading = lsm303_heading();
		HC05_send(heading);
	}
}
```