Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

Sistemi daljinskega vodenja

Programska in strojna oprema za komunikacijo in vodenje quadcoptera

Avtor: Mario Gavran E5029604

Program in smer: ELEKTRONIKA

Letnik: 2

Maribor, Junij 2020

Vsebina

Uvod

Ta projekt je poskus izdelave preprostega, ampak funkcionalen quadcopterja, narejenega z uporabo poceni in razpoložljivih komponent. Namen projekta je prikazati princip delovanja enostavnega quadcopterja, njegovega regulatora in brezžično komunikacijo za krmiljenje vozila. Torej bo tukaj prikazan primer zglednega vozila in programske opreme za vozilo in daljinski upravljanik.

Obstajata dva osnovna načina upravljanja quadcoptera. Prvi, enostavnejši način, je regulacija hitrosti rotacije okoli 3 kartezijske osi vozila, in drugi, bolj zapleten, pa je regulacija kota okoli tistih osi. V tem dokumentu je izdelan quadcopter s prvo, enostavnejšo različico regulacije, regulacijo kotne hitrosti vozila okoli 3 osi. Pilot ki upravlja quadcopter, bo uravnaval hitrost rotacije okrog osi, ne pa absolutni kot vozila okoli tiste osi. Ta način regulacije otežuje krmiljenje quadcoptera, saj ne obstaja programska oprema ki bi uravnavala kot vozila, temveč le hitrost rotacije vozila. Kar pomeni da mora pilot določiti želeni kot vozila preprosto s spreminjanjem želene hitrosti rotacije. Ko vozilo doseže želeni kot, pilot nastavi hitrost rotacije vozila na nič. Nato regulatorji hitrosti rotacije vozila poskušajo ohraniti hitrost rotacije na ničli in s tem obdrže nastavljeni kot vozila.

Ta quadcopter je sestavljen iz naslednjih delov:

- 4 BLDC Motorja
- 4 krmilnika hitrosti BLDC motorja(ESC)
- Žiroskop
- 2 mikrokrmilnika
- 2 xy-joystick-potenciometra
- par modulov 433RF OOK za brezžično komunikacijo

Krmilniki hitrosti(ESC), RF moduli in žiroskop so kupljeni, a mikrokrmilniki so na razvojnih ploščah sa že narejenim napajalnikom.

Regulatorji hitrosti rotacije vozila, algoritmi krmiljenja letenja in komunikacijski vmesnik za sprejem podatkov od daljinskega upravljalnika so del programske in strojne opreme mikrokrmilnika na vozilu.

Vzorčenje in pošiljanje ukazov od pilota so del programske in strojne opreme mikrokrmilnika na daljinskem upravljalniku.

Opis upravljanja gibanja quadcoptera

Quadcopter je multirotor vozilo s 4 motorji razporejeni na vogalih vozila. Vozilo upravlja pilot z zagotavljanjem ukazov za želeno gibanje vozila. Pilot pošilja želene hitrosti rotacije okoli 3 kartezijeve osi, X, Y in Z, a to dela z dvema XY joysticka na daljinskem upravljalniku. Vsak joystick ima dva potenciometra. En joystick je namenjen za gas in Z os, drugi pa za X in Y osi. Spreminjanjem položaja joysticka, daljinski upravljalnik pošlje vozilu prek brezžične komunikacije želeno hitrost rotacije okoli določene osi, in nato pa vozilo poskuša kar bolje slediti pilotove želene hitrosti.

Slika 1 prikazuje osnovno skico quadcoptera z motorji in propelerji, z označenimi X, Y in Z osmi okoli katerih se quadcopter vrti in z označeno smerjo vrtenja posameznega motorja. Če povečamo hitrost vrtenja dveh levih motorjev in zmanjšamo hitrost dveh desnih motorjev, se bo quadcopter začel rotirati v desno.Če povečamo ali zmanjšamo hitrost dveh prednjih ali dveh zadnjih motorjev, se bo quadcopter nagibal naprej ali nazaj. Če spremenimo hitrost diagonalnih motorjev, povečamo hitrost motorja na eni diagonali in jo zmanjšamo na drugi, se bo quadcopter vrtil okoli tretje, Z osi. Če združimo vse tri načine, lahko premikamo vozilo v katerokoli smer. Če hkrati povečamo hitrost vseh štirih motorjev, se quadcopter dvigne v višino.

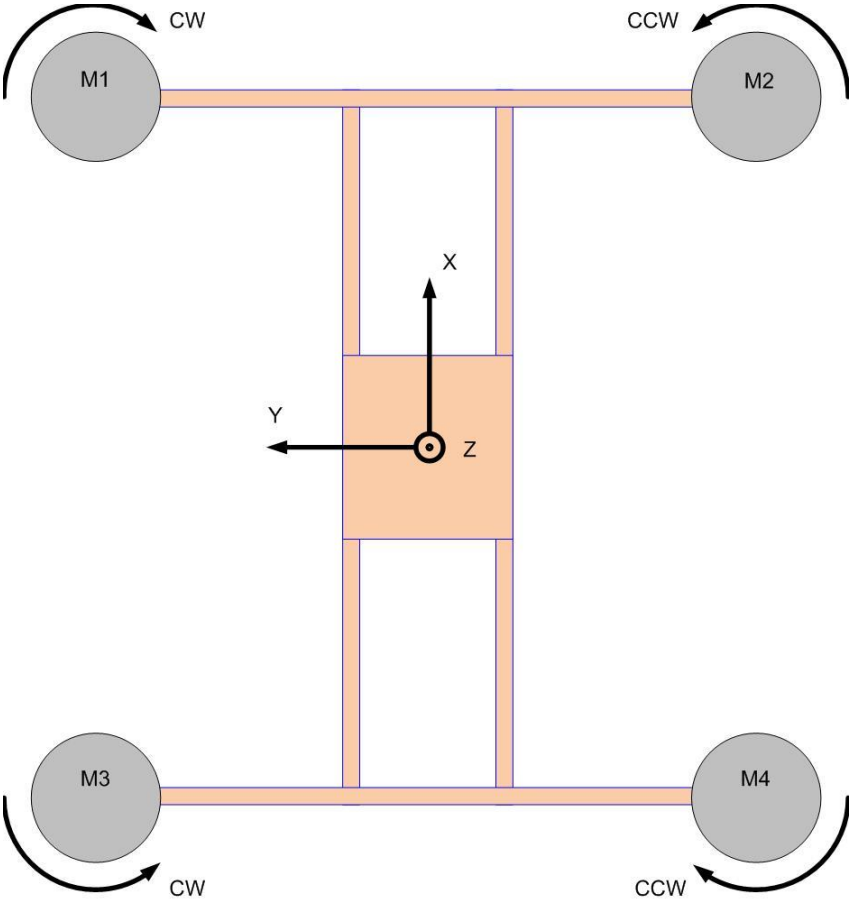


Figure 1

Opis sistema

Zgoraj navedene zahteve za krmiljenje gibanja vozila ne zadoščajo za stabilno vzdrževanje vozila v zraku. Zaradi neenakomerne moči in navora motorja, nepopolne konstrukcije in nepravilno porazdeljene mase samega vozila, ni dovolj, da se hitrost vseh motorjev enakomerno poveča. Močnejši motorji zavrtijo vozilo tako, da se dvignejo hitreje kot drugi, šibkejši motorji, in neuravnovešena masa vozila bo gravitirala s težke strani vozila. Zato je za vzdrževanje vozila na želeni višini potreban regulator s integralnim delovanjem za vsako posamezno os vozila, ki bo vzdrževal vozilo tako da bo šibkejši motorji in težje strane vozila imale več moči za vzpon. Zaradi same narave letala, ki je zelo nestabilna, potreben je tudi regulator s derivacijskim delovanjem, ki bo vozilu dodal potrebno inercijo in tako olajšal krmiljenje vozila. Zaradi potrebe po spremembi inteziteta reakcije quadcoptera je potreben tudi sorazmerni regulator.

Quadcopter nosi mikrokrmilnik, ki izvaja programsko opremo za letenje. Ta program prebere trenutne hitrosti rotacije vseh treh osi vozila z žiroskop modulom, nameščenim na sredino vozila, in sprejema pilotne ukaze preko brezžične komunikacije, preko antenskega sprejemnega modula, ki je tudi na vozilu.

Žiroskop je priključen na I2C periferno vodilo mikrokrmilnika, antenski modul pa na periferijo UART. Pilot ima daljinski upravljalnik, ki s pomočjo drugega mikrokrmilnika prebere stanje obeh xy-joystickov in jih pošlje preko periferije UART na oddajni antenski modul. Vsaka os posameznega xy-joysticka predstavlja eno od 4 željena parametra, gas(Throttle) in 3 rotacije, x(Roll), y(Pitch) i z(Yaw).

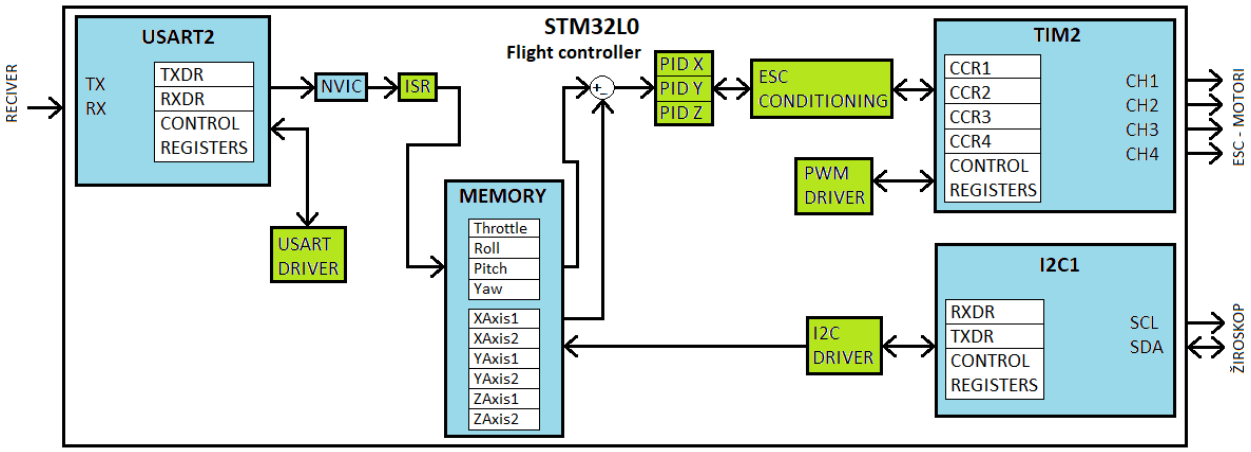


Figure 2

Slika 2 prikazuje proces upravljanja quadcoptera. Ta shema je predstava programske oreme ki se izvaja na mikrokrmilniku na vozilu. Na sliki so strojni periferni bloki označeni z modro barvo, in programski bloki z zeleno barvo.

Na sliki so prikazana 3 PID regulatorja, en regulator za vsako os vozila. Mikrokrmilnik sprejema želeno vrednost hitrosti rotacije vozila z daljinskega upravljalnika prek sprejemnega modula in perifernih blokov USART in NVIC. Komunikacijski protokol je razložen kasneje.

Podatki žiroskopa in daljinskega upravljalnika so shranjeni v pomnilniku in jih regulator po potrebi vzame. Z vrstnim redom izvajanja programa je zagotovljeno da so podatki iz žiroskopa in joystickov, ki jih

sprejeme regulator, iz enakega časovnega intervala. Timer TIM2 se uporablja za generiranje PWM signalov, ki služijo za krmiljenje regulatorjev hitrosti motorja(ESC). Tako lahko samo regulator neposredno krmili hitrost motorja. Pilot samo pošilja želene hitrosti rotacije kot želeni parameter, regulator pa upošteva razliko med želenim parametrom in podatkom iz žiroskopa kot vhodno vrednost v regulator. Tista vrednost se uporablja za izračun izhodne vrednosti regulatora.

Če pilot ne želi nobenega premika vozila, nastavi joysticke v nevtralne položaje in tako nenehno pošilja ničle na sprejemni antenski modul, za vse 3 osi. To pomeni, da bo regulator prejemal podatke iz žiroskopa odštete od ničel prejetih iz sprejemnega antenskega modula kot vhodne podatke, in na ta način prepreči gibanje vozila. Na primer, če se vozilo začne vrteti v eno stran, žiroskop to opazi in nato regulator na vhod prejeme tisto vrednost iz žiroskopa odšteto od ničle iz antenskega sprejemnega modula. Regulator začne spreminjati svoje izhodne podatke tako da čim prej ustavi vrtenje vozila in na vhodu v regulator prejeme nič.

S spremembo želenega parametra na eni osi joysticka, pilot dodeli vozilu, da se zavrti okoli želene osi z želeno hitrostjo. Regulator pa spremeni krmilne signale za ESC, s čimer doseže želena hitrost rotacije vozila. Ko so podatki iz žiroskopa enaki kot želena vrednost vhodna vrednost v regulator je nič kar pomeni da je dosežena želena hitrost rotacije vozila.

Daljinska komunikacija

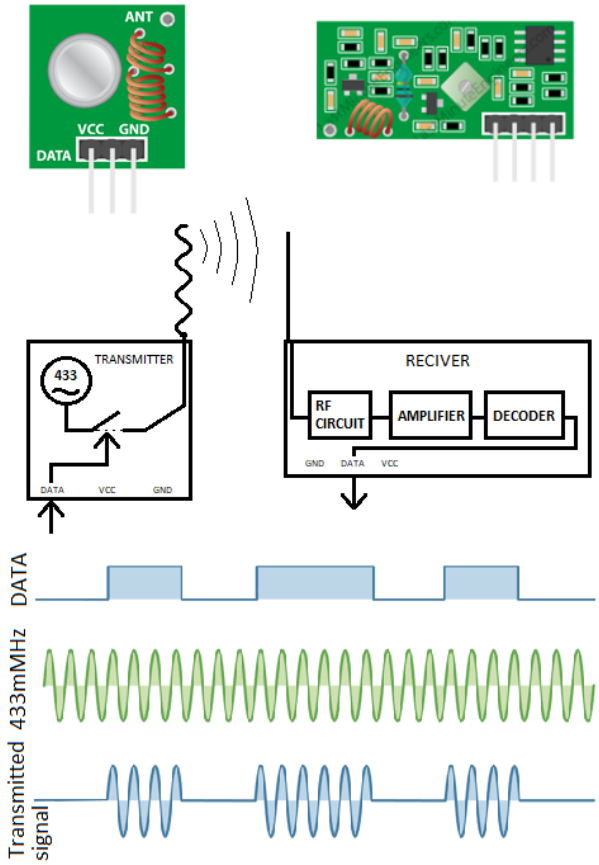


Figure 3

Fizična plast daljinske komunikacije tega quadcoptera je sestavljena iz 433MHz OOK modula in perifernega UART modula v mikrokrmilniku. Kot pravi ime modula, gre se za on/off keying modulacijo na 433MHz. Bit 0 bo kodiran, kot da ni signala (off) in bit 1 kot da obstaja signal (on). Tako oddajnik oddaja signal samo za bitove 1 in ne oddaja signala za bitove 0. Ta vrsta kodiranja se lahko razume kot ASK, kjer je bit 0 signal amlitude 0, oziroma ni signala, in bit 1 signal amplitude 1 ali katera koli amplituda različna od 0.

Na sprejemnem antenskem modulu je ojačevalnik s spremenljivim ojačanjem. Ta ojačevalnik bo začel povečevati ojačanje če trenutno ne prejema signal od 433MHz, in ga bo še naprej povečeval dokler ne prejeme tak signal. To pomeni, da če oddajnik ne pošilja podatke ali pošilja samo ničle, bo sprejemni antenski modul začel povečevati ojačenje ojačevalnika, dokler ne prejeme signal od 433MHz in zato bo lahko začel sprejemati šum kot signal ali signal drugega oddajnika na 433MHz. Zaradi te pomanjkljivosti sprejemnika je obseg oddajnika zmanjšan na desetak metrov in določa potrebo po neprekinjenem prenosu podatkov, da ne bi ojačevalnik začel povečavati ojačanje in s tem okvari prejete podatke. Kot komunikacijski vmesnik se uporablja periferija USART v mikrokrmilniku. Antenski moduli so neposredno povezani z TX in RX pine.

Throttle_H	X X X X X	0 0 0
Throttle_L	X X X X X	0 0 1
XAxis_H	X X X X X	0 1 0
XAxis_L	X X X X X	0 1 1
YAxis_H	X X X X X	1 0 0
YAxis_L	X X X X X	1 0 1
ZAxis_H	X X X X X	1 1 0
ZAxis_L	X X X X X	1 1 1

Sprejemnik začne motiti obliko dekodiranega diskretnega signala, če frekvenca spremembe stanja presega 2,5 kHz. Natančneje, sprejemnik začne zožiti visoko in razširiti nizko stanje nad frekvenco od 2,5kHz, kar pomeni, da bo periferna naprava USART začela napačno brati podatke, ki jih prejme. Zaradi tega je izbrana hitrost 2400bps za vmesnik USART.

Slika 4 prikazuje predstavo programske oreme za daljinski upravljalnik. Analogno-digitalni pretvornik v mikrokrmilniku periodično vzorči napetost na 4 potenciometra pri 10-bitni ločljivosti in jih shrani v pomnilnik. Po vsakem branju AD pretvornika, se vsaka vrednost od 10 bitov razdeli na dve vrednosti od 5 bitov, visoke in nizke dele. Na vsak od teh 5 bitov je dodano še 3 bitov, ki predstavljajo zaporedno število od 000 do 111. Tako se pošlje 8 bajtov za

vsa 4 potenciometra, vsak 2 bajta. Vsaka 2 bajta ki sta prejeta ena za drugim, predstavljata eno vrednost od 10 bitov. Sprejemnik sprejema sporočila prek USART in NVIC periferije. Vsakič, ko periferija USART prejme en bajt, NVIC generira prekinitev in sproži prekinitveno rutino, v kateri je prejeti bajt najprej

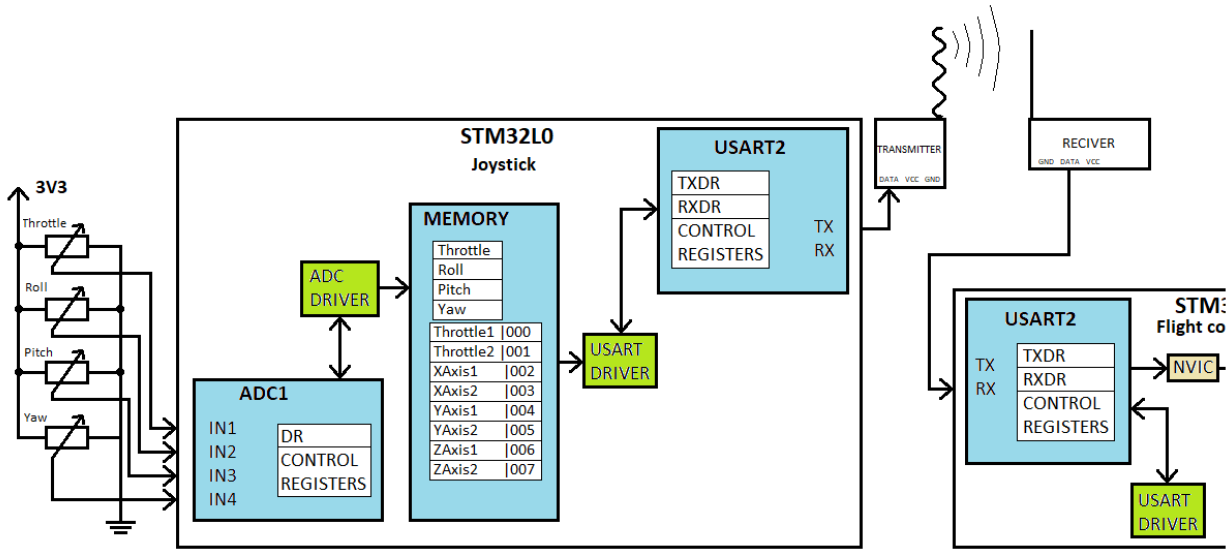


Figure 4

preverjen z zaporedno številko. Če je prejeta zaporedna številka bajta v redu, se komunikacija nadaljuje. Če je prejetih vseh 8 bajtov v vrsti, je celoten paket shranjen v pomnilniku in je na voljo regulatoru. Če katerikoli od bajtov ni prispel v vrsti, je celoten paket z 8 bajtov zanemarjen, in se čaka na bajt z zaporedno številko 000. Tako je zagotovljeno, da so 10 bitne vrednosti joysticka iz enakega časa vzorčenja in da se kasneje lahko razdeljeni bajti pravilno združijo v 10-bitne vrednosti. Program prekinitvene rutine je prikazan na naslednji strani na sliki 6.

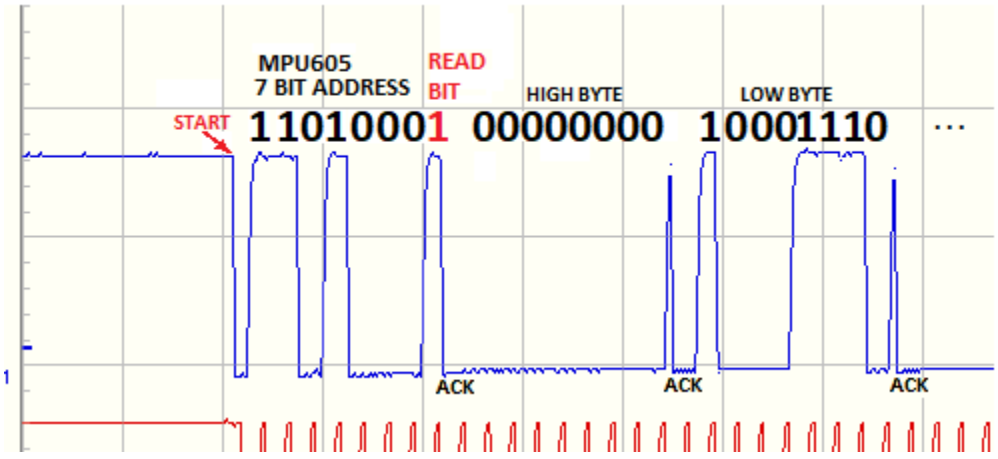


Figure 5

Slika 5 prikazuje časovni potek I2C signala. Prvi bajt je I2C naslov žiroskopa, druga dva pa visoki in nizki bajt X osi.

```
void USART2_IRQHandler(void)
{
  while (!(USART2->ISR & USART_ISR_RXNE));
  uint8_t RecivedByte = USART2->RDR;

  static uint8_t Tag = 0x00U;
  if((RecivedByte & 0x07U) == 0x00) Tag = 0x00;     // 0x07 = 0b00000111

  if((RecivedByte & 0x07U) == Tag)
  {
    switch((RecivedByte & 0x07U))
    {
    case 0x00:    // XXXXX000
      Tag=0x01U;
      JoystickTP->Throttle  = ((0xF8U & RecivedByte)<<2);
      break;
    case 0x01:    // XXXXX001
      Tag=0x02U;
      JoystickTP->Throttle  |= ((0xF8U & RecivedByte)>>3);
      break;
    case 0x02:    // XXXXX010
      Tag=0x03U;
      JoystickTP->Roll    = ((0xF8U & RecivedByte)<<2);
      break;
    case 0x03:    // XXXXX011
      Tag=0x04U;
      JoystickTP->Roll    |= ((0xF8U & RecivedByte)>>3);
      break;
    case 0x04:    // XXXXX100
      Tag=0x05U;
      JoystickTP->Pitch     = ((0xF8U & RecivedByte)<<2);
      break;
    case 0x05:    // XXXXX101
      Tag=0x06U;
      JoystickTP->Pitch     |= ((0xF8U & RecivedByte)>>3);
      break;
    case 0x06:    // XXXXX110
      Tag=0x07U;
      JoystickTP->Yaw       = 0; //((0xF8U & RecivedByte)<<2);
      break;
    case 0x07:    // XXXXX111
      Tag=0x00U;
      JoystickTP->Yaw       |= 0; //((0xF8U & RecivedByte)>>3);
      *JoystickTP1 = *(JoystickTP);
      break;
    default:
      break;
    }
  }
}
```

Figure 6

Programska oprema

Zaradi enostavnosti razlaganja programske kode je prikazan samo en regulator in samo ena os, ostale osi in regulatori so podobni. Na začetku programa so deklarirane funkcije, spremenljivke in podatkovne strukture, potem so definirani parametri regulatorja. Main funkcija se začne z inicializacijo uporabljenih perifernih strojnih blokov, TIM2, I2C1 in USART2 periferne naprave. Inicializacija USART periferije omogoča tudi sprejem prekinitev, kar pomeni, da program sprejema podatke iz daljinskega upravljalnika, čeprav regulator še ni zagnan.

```c
typedef struct
{
  uint16_t Throttle;
  uint16_t Roll;
  uint16_t Pitch;
  uint16_t Yaw;
}JOY_DATA_Type;


typedef struct
{
  uint8_t XAxisH;
  uint8_t XAxisL;
  uint8_t YAxisH;
  uint8_t YAxisL;
  uint8_t ZAxisH;
  uint8_t ZAxisL;
}GYRO_DATA_Type;
```

```c
#include "stm32l0xx.h"                  // Device header.
#include "USART2_STM32L031K6.h"         // USART driver.
#include "PWM_TIM2.h"                   // PWM driver.
#include "I2C1_STM32L031K6.h"           // I2C driver.
#include <stdint.h>                     // Standard integer data types.
#include "My_Data_Types.h"              // Data structures.


//********* INTERUPT HANDLER DECLARATION ************************************************
void USART2_IRQHandler(void);


//********* GLOBAL DATA STRUCTURES INSTANCES: ***************************************
JOY_DATA_Type   JoystickBuffer;                     // Reciver buffer.
JOY_DATA_Type*  JoystickBufferP = &JoystickBuffer;

JOY_DATA_Type   Joystick;                           // Reciver data.
JOY_DATA_Type*  JoystickP = &Joystick;

GYRO_DATA_Type  Gyroscope;                          // Gyroscope data.
GYRO_DATA_Type* GyroscopeP = &Gyroscope;
```

```c
//********* PID CONTROLLER GAIN SETTINGS: ***********************************************
#define   P_X_GAIN  1
#define   I_X_GAIN  0.005
#define   D_X_GAIN  0.5
```

```c
//******************************** main() FUNCTION: ***********************************
int main(void)
{

  PWM_TIM2_init();                          // Initialize PWM
  initI2C1(MPU6050Address);                 // Initialize I2C
  initUSART2();                             // Initialize USART2


//******************************** MAIN-SCOPE VARIABLES: ******************************
  // Throttle variables:
  int16_t   Throttle_Data=0;                // For conditioning reciver throttle data.
  uint32_t  Throttle_Offset=0;              // Initial throttle offset.

  // Roll (X) axis gyroscope & reciver variables:
  int16_t   Roll_Data=0;                    // For conditioning reciver Roll data.
  int32_t   Roll_Offset=0;                  // Recivers initial Roll offset.
  int16_t   GYRO_X_Data=0;                  // For conditioning gyroscope X data.
  int32_t   GYRO_X_Offset=0;                // Initial gyroscope X offset.
```

```c
  // Roll (X) PID Controller variables:
  int16_t   PID_X_OUT=0;                    // PID regulator output
  int16_t   I_X_OUT=0;                      // Intergral regulator output
  int16_t   ERR_X=0;                        // Current rror
  int16_t   PERR_X=0;                       // Previous error
```

```c
  // ESC PWM CCR temporary variables:
  int16_t   ESC_CCR1=0;
  int16_t   ESC_CCR2=0;
  int16_t   ESC_CCR3=0;
  int16_t   ESC_CCR4=0;
```

Potem se zažene konfiguracija žiroskopa, kjer se želeni podatki vnesejo v registre za konfiguracijo žiroskopa preko vodila I2C, ki je bil prvotno inicializiran. Mikrokrmilnik ponavlja vnos istih vrednosti v registre, dokler iz istih registrov ne prebere, kaj je vnesel, in s tem program zagotavlja da je komunikacija potekla brez napak in da je žiroskop pravilno konfiguriran.Takoj po koncu konfiguracije se določijo odmiki/offseti žiroskopa.

```
93  //****************************************** MPU6050 setup    ************************************************
94    do                                                          // Make sure that MPU6050 registers are set correctly.
95    {
96      I2C1_writeData(0x1B,0x18);                                // MPU6050 REG:Gyroscope Config, Full scale FS_SEL[1:0]=2
97      I2C1_writeData(0x6B,0x00);                                // MPU6050 REG:Power Management1, Disable sleep mode
98    }while( !(I2C1_readData(0x1B)==0x18) ||
99            !(I2C1_readData(0x6B)==0x00) );                     // Repeat until you read what you wrote.
100
101
102 //****************************************** GYROSCOPE offset ************************************************
103   while(i<3000)                                                       // Acquire 3000 samples and sum them all.
104   {
105     I2C1_readMulData(0x43,6U,GyroscopeP);                             // Read all axies.
106     GYRO_X_Offset += (int16_t)(0xFFFF & (((GyroscopeP->XAxisH) << 8) |
107                                (GyroscopeP->XAxisL)));
108     GYRO_Y_Offset += (int16_t)(0xFFFF & (((GyroscopeP->YAxisH) << 8) |
109                                (GyroscopeP->YAxisL)));
110     GYRO_Z_Offset += (int16_t)(0xFFFF & (((GyroscopeP->ZAxisH) << 8) |
111                                (GyroscopeP->ZAxisL)));
112     i++;                                                            // Increment counter.
113   }
114   GYRO_X_Offset /= 3000;                                            // Calculate offsets.
115   GYRO_Y_Offset /= 3000;
116   GYRO_Z_Offset /= 3000;
117   i=0;                                                              // Reset counter to be used for JOYSTICK offset
```

Ker je prekinitvena rutina že omogočena, so podatki, prejeti z daljinskega upravljalnika, že shranjeni v pomnilniku in se nenehno osvežujejo. Joystick potenciometer za menjavo plina ima nevtralni položaj na skrajnem spodnjem položaju, medtem joystick potenciometri za regulacijo hitrosti rotacije vozila imajo nevtralni položaj v sredini, kar nam omogoča da nižje položaje interpretiramo kot negativne in višje položaje kot pozitivne. Torej moramo določiti odmik/offset vseh krmilnih joystick potenciometrov v nevtralnem položaju, ki se nato odšteje od vsake nove vrednosti ki jo prejmemo od daljinskega upravljanika. Določanje odmika ali offseta se začne ko je potenciometer za gas v najnižjem položaju, kar zagotavlja da se ločljivost gasa ne zmanjša če bi se nenamerno joystick gasa zadržao v višjem položaju. Gas ne more biti negativen, ker je kasneje omejen na 0.

```
120 //****************************************** JOYSTICK offset ************************************************
121   while( ((JoystickP->Throttle)<<1) > 200)                          // Wait until stick is in low position.
122   {
123     TIM2->CCR1 = 2200 + ((JoystickP->Throttle)<<1);                 // While waitnig, update the PWM CCR registers.
124     TIM2->CCR2 = TIM2->CCR1;
125     TIM2->CCR3 = TIM2->CCR1;
126     TIM2->CCR4 = TIM2->CCR1;
127   }
128
129   while(i<2000)                                                       // Acquire 2000 samples.
130   {
131     Throttle_Offset += (int16_t)(0x07FFU & ((JoystickP->Throttle)<<1) );    // Sum all reads.
132     Roll_Offset     += (int16_t)(0x07FFU & (JoystickP->Roll));
133     Pitch_Offset    += (int16_t)(0x07FFU & (JoystickP->Pitch));
134     Yaw_Offset      += (int16_t)(0x07FFU & (JoystickP->Yaw));
135     i++;                                                              // Increment counter.
136   }
137   Throttle_Offset /= 2000;                                            // Calculate offsets.
138   Roll_Offset     /= 2000;
139   Pitch_Offset    /= 2000;
140   Yaw_Offset      /= 2000;
```

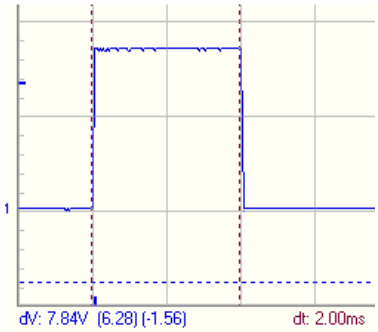
Po inicializaciji strojne opreme in določitvi odmika se zažene regulator. Najprej prebere najnovejše podatke iz žiroskopa in shrani v pomnilnik. Komunikacija med procesorjem in žiroskopom poteka preko I2C periferije, ki je definirana za pošiljanje 8 bitov(1bajt), in dolžina besede za vsako os žiroskopa je 16 bitov(2bajta). Ker komunikacija poteka z 8 bitov, se pošljejo prvo visoki bajt in nato nizki bajt. Potem se združe se oba bajta v eno besedo od 16 bitov in se tako uporablja. Od vseh vrednosti se odšteje offset in določi območje neobčutljivosti.

```
146     // Read all 3 axis from the gyroscope.
147     I2C1_readMulData(0x43,6U,GyroscopeP);
148
149     // Gyroscope data conditioning:
150     GYRO_X_Data = (int16_t)(((int16_t)(0xFFFF & ( ((GyroscopeP->XAxisH)<<8) | (GyroscopeP->XAxisL))) - GYRO_X_Offset)/5);
151     if(((int16_t)(GYRO_X_Data)<(2)) && ((int16_t)(GYRO_X_Data)>(-2))) GYRO_X_Data=0;
152
153     GYRO_Y_Data = (int16_t)(((int16_t)(0xFFFF & ( ((GyroscopeP->YAxisH)<<8) | (GyroscopeP->YAxisL))) - GYRO_Y_Offset)/5);
154     if(((int16_t)(GYRO_Y_Data)<(2)) && ((int16_t)(GYRO_Y_Data)>(-2))) GYRO_Y_Data=0;
155
156     GYRO_Z_Data = (int16_t)(((int16_t)(0xFFFF & ( ((GyroscopeP->ZAxisH)<<8) | (GyroscopeP->ZAxisL))) - GYRO_Z_Offset)/5);
157     if(((int16_t)(GYRO_Z_Data)<(2)) && ((int16_t)(GYRO_Z_Data)>(-2))) GYRO_Z_Data=0;
158
159
160     // Reciver data conditioning:
161     NVIC_DisableIRQ(USART2_IRQn);                                          // Disable interupt while conditioning
162     Throttle_Data = (0x07FFU & ((JoystickP->Throttle)<<1))- Throttle_Offset;        //    (0) - (+2046)
163
164     Roll_Data   = (int16_t) ((0x03FFU & ((JoystickP->Roll)))  - Roll_Offset);       // (-512) - (+511)
165     if(((int16_t)(Roll_Data)<(11)) && ((int16_t)(Roll_Data)>(-11))) Roll_Data=0;    // Deadband for Roll data.
166
167     Pitch_Data  = (int16_t) ((0x03FFU & ((JoystickP->Pitch))) - Pitch_Offset);      // (-512) - (+511)
168     if(((int16_t)(Pitch_Data)<(11)) && ((int16_t)(Pitch_Data)>(-11))) Pitch_Data=0; // Deadband for Pitch data.
169
170     Yaw_Data    = (int16_t) ((0x03FFU & ((JoystickP->Yaw)))   - Yaw_Offset);       // (-512) - (+511)
171     if(((int16_t)(Yaw_Data)<(11)) && ((int16_t)(Yaw_Data)>(-11))) Yaw_Data=0;      // Deadband for Pitch data.
172     NVIC_EnableIRQ(USART2_IRQn);                                           // Enable interupt
173
174
175     // PID Roll (X) controller calculation:
176     ERR_X     = GYRO_X_Data - Roll_Data;                        // Current error.
177     I_X_OUT   = I_X_OUT + ERR_X*I_X_GAIN;                       // Integral controller output.
178     PID_X_OUT = ERR_X*P_X_GAIN + I_X_OUT + (PERR_X-ERR_X)*D_X_GAIN;  // PID controller output.
179     PERR_X    = ERR_X;                                          // Previous error for next controller loop(initially is 0).
```

Prekinitve so onemogočene pred pripravo podatkov s joysticka, in so ponovno omogočene po zaključku priprave podatkov. Podatek za gas je brez predznaka in se premakne za en bit v levo, kar pomeni da se poveča dolžina podatka za gas na 11 bitov(2048). To je pomembno zato ker je dolžina podatka, ki ga prejeme TIM2, za generiranje PWM signalov, tudi dolg 11 bitov (2048). Na ta način lahko uporabljamo celoten raspon hitrosti motora samo z joystickom za gas. Ostali podatki, Roll, Pitch in Yaw so podatki sa predznakom in so omejeni na 10 bitov, kar pomeni da imajo obseg od -512 do +511. Od vseh podatkov se odšteje offset in določi območje neobčutljivosti.

Nato se začne izračun PID regulatorja in sprememba signala za regulacijo hitrosti motorja. Priložena je celotna izvorna koda, tukaj so prikazani samo odrezki.

Signali za regulator hitrosti motora(ESC) se generirajo s pomočjo časovnika TIM2. Časovnik ustvari PWM signal frekvence 100 Hz in informacije se kodirajo v delovnem ciklu signala. Najmanjša dolžina visokega stanja signala je 1 ms, največja dolžina 2 ms, perioda signala je vedno 10ms(100Hz). To pomeni, da je hitrost osveževanja motorja 10ms(100Hz), saj bo ESC spremenil hitrost motorja samo pri spremembah delovneg cikla. Nastavitve časovnika so izbrane tako, da je celotno območje dolžine signala predstavljeno z 11 bitov ali 2048, ker je ločljivost gasa tudi 11 bitov.



Regulator

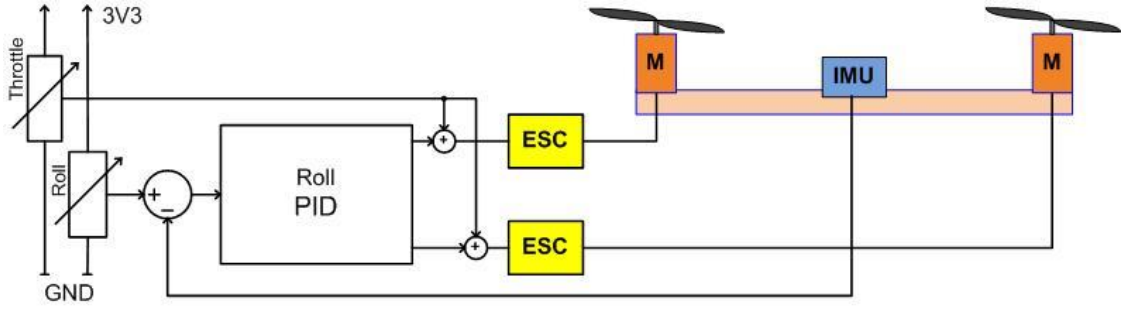


Figure 7

Slika 7 prikazuje poenostavljen diagram regulacijskog kroga. Zaradi enostavnosti razlaganja je prikazan samo en regulator in samo ena os vozila. Pri izračunu signala za ESC se upoštevajo vsi regulatorji, tako da se seštejejo vsi prispevki posameznih regulatorjev za motor na tem ESCju. Še se doda velikost gasa enako za vse motorje.

Regulatorji hitrosti motorjev(ESC) delujejo kot izvršni člani, žiroskop pa je merilni član. Regulator najprej izračuna regulacijsko napako, tj. razliko med želeno in izmerjeno vrednostjo. Nato izračuna integracijski del, tako da v spremenljivku za integracijski del regulatora sešteje novo vrednost produkta integracijske konstante in trenutne regulacijske napake (I_OUT = I_OUT + ERR*I_GAIN). Če je regulacijska napaka enaka nič pomeni da je dosežena želena vrednost in da se v integracijski del seštejo le ničle, oziroma, izhod iz regulatora se ne spreminja.

```
175     // PID Roll (X) controller calculation:
176     ERR_X     = GYRO_X_Data - Roll_Data;
177     I_X_OUT   = I_X_OUT + ERR_X*I_X_GAIN;
178     PID_X_OUT = ERR_X*P_X_GAIN + I_X_OUT + (PERR_X-ERR_X)*D_X_GAIN;
179     PERR_X    = ERR_X;
```

```
211     // Conditioning TIMERs PWM signal high-time duration:
212     ESC_CCR1 = 2100 + (Throttle_Data + PID_X_OUT + PID_Y_OUT + PID_Z_OUT);
213     ESC_CCR2 = 2100 + (Throttle_Data - PID_X_OUT + PID_Y_OUT - PID_Z_OUT);
214     ESC_CCR3 = 2100 + (Throttle_Data + PID_X_OUT - PID_Y_OUT - PID_Z_OUT);
215     ESC_CCR4 = 2100 + (Throttle_Data - PID_X_OUT - PID_Y_OUT + PID_Z_OUT);
```

```
231     // Update PWM CCR registers
232     TIM2->CCR1 = ESC_CCR1;
233     TIM2->CCR2 = ESC_CCR2;
234     TIM2->CCR3 = ESC_CCR3;
235     TIM2->CCR4 = ESC_CCR4;
```

Nato se izračuna derivacijski del tako da množi derivacijsko konstanto z razliko med trenutno in prejšnjo regulacijsko napako. Torej, izhod iz derivacijskog regulatora obstaja le če se regulacijska napaka spreminja. Na primer, če vozilo miruje in pilot pošilja samo ničle pomeni da je regulacijska napaka enaka nič. Potem se vozilo nenadoma začne vrteti v desno okoli X osi, medtem ko pilot še vedno pošilja samo ničle, kar pomeni da zdaj obstaja regulacijska napaka in derivacijski regulator se bo odzval. Če se vozilo še naprej vrti v isti smeri z enako hitrostjo, bo regulacijska napaka ostala nespremenjena in razlika med

trenutno in preteklo regulacijsko napako bo nič, kar pomeni da se derivacijski regulator odziva le na začetku spremembe hitrosti rotacije.

Slika 8 prikazuje odziv regulatora X osi. Mikrokrmilnik, po izračunu vseh regulatorjev, pošlje vrednost izhoda regulatora X osi, prek USART periferije, programatora in USB kabla na osebni računalnik, kjer se podatki shranjujejo. Med testiranjem je bilo vozilo obešeno na dveh točkah vzdolž X osi ker mora biti ves čas povezanz računalnikom prek USB kabla.

Slika prikazuje 3 ločene dogodke, oziroma 3 ločene serije nihanj. Prva nihanja so se pojavila ko se je velikost gasa spremenila z 0 na približno 10%. To se je zgodilo zaradi nepopolno razporejene mase vozila, razlik v motorjih in drugih dejavnikov. Po nekaj sekundah se nihanja ustavijo in vozilo se stabilizira. Drugi dve nihanji se nista pojavili zaradi spremembe velikosti gasa, ampak zaradi zunanjih motenj. Prvo od teh dveh nihanj je povzročil šibek udarec v eno stran vozila. Po ponovni stabilizaciji vozila je drugi niz nihanj povzročil nekoliko močnejši udarec na drugo stran vozila.

Rezultati testa niso povsem zanesljivi, saj je bilo vozilo obešeno na dveh točkah, kar ni enako kot ko vozilo prosto leti po zraku. Ko vozilo prosto leti v zraku, pričakujemo različne rezultate.

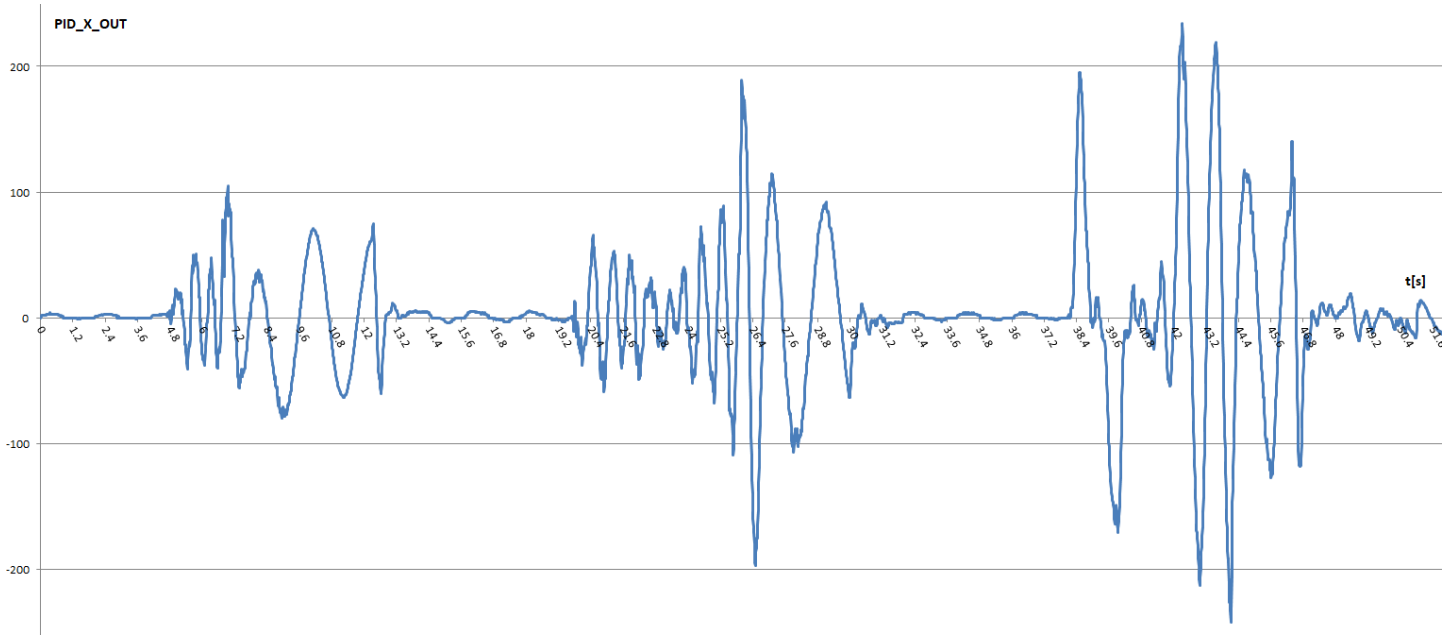


Figure 8

Uporabljeni deli

BLDC motor

Ime motora:	A2212/13T
Koeficient napetosti:	930
Tok v prostem teku:	0.5 A pri 10 V
Nazivni tok:	10 A
Največji tok:	13 A
Največja snaga:	150 W
Masa:	52,7 g

Krmilnik hitrosi BLDC motora

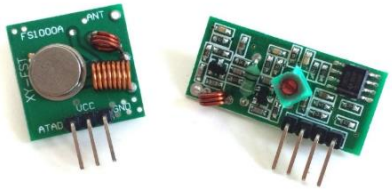
Ime krmilnika:	ReadytoskyESC
Nazivni tok:	30 A
Največji tok:	35 A (10 s)
Masa:	32 g
Hitrost osveževanja:	100 Hz

IMU/Žiroskop

Ime modula:	MPU-6050
Hitrost osveževanja:	8kHz
Območje(pospešek):	±2g, ±4g, ±8g, ±16g
Območje(hitrost rot.):	±250, ±500, ±1000, ±2000°/s

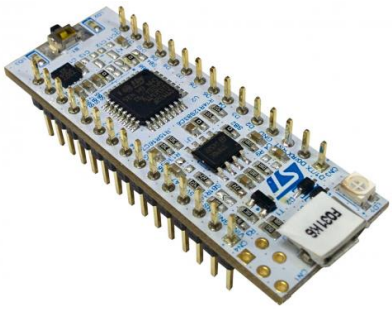
RF moduli

Napetost(oddajnik):	3-12V
Napetost(sprejemnik):	5V
Tok oddajnika:	30mA
Največja hitrost prenosa:	2,5kHz



Mikrokrmilnik

Ime mikrokrmilnika:	STM32L031K6
Flash:	32k
RAM	8k
CPU frekvenca	2,1MHz



Električna shema