Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

INDUSTRIA INGENIARITZA TEKNIKOKO ATALA

SECCIÓN INGENIERÍA TÉCNICA INDUSTRIAL

INDUSTRIA ELEKTRONIKAREN ETA AUTOMATIKAREN INGENIARITZAKO GRADUA

GRADU AMAIERAKO LANA

2016 / 2017

ERAIKIN BATEN BARRUALDEAREN KARTOGRAFIA EGITEKO AUTOA

III.DISEINUA

IKASLEAREN DATUAK	**ZUZENDARIAREN DATUAK**
IZENA DAVID	IZENA OSKAR
ABIZENAK ARAMBURU RAMOS	ABIZENAK CASQUERO OYARZABAL
	SAILA SISTEMEN INGENIARITZA ETA AUTOMATIKA SAILA
Sinadura	Sinadura
DATA 2017-04-22	DATA 2017-04-27

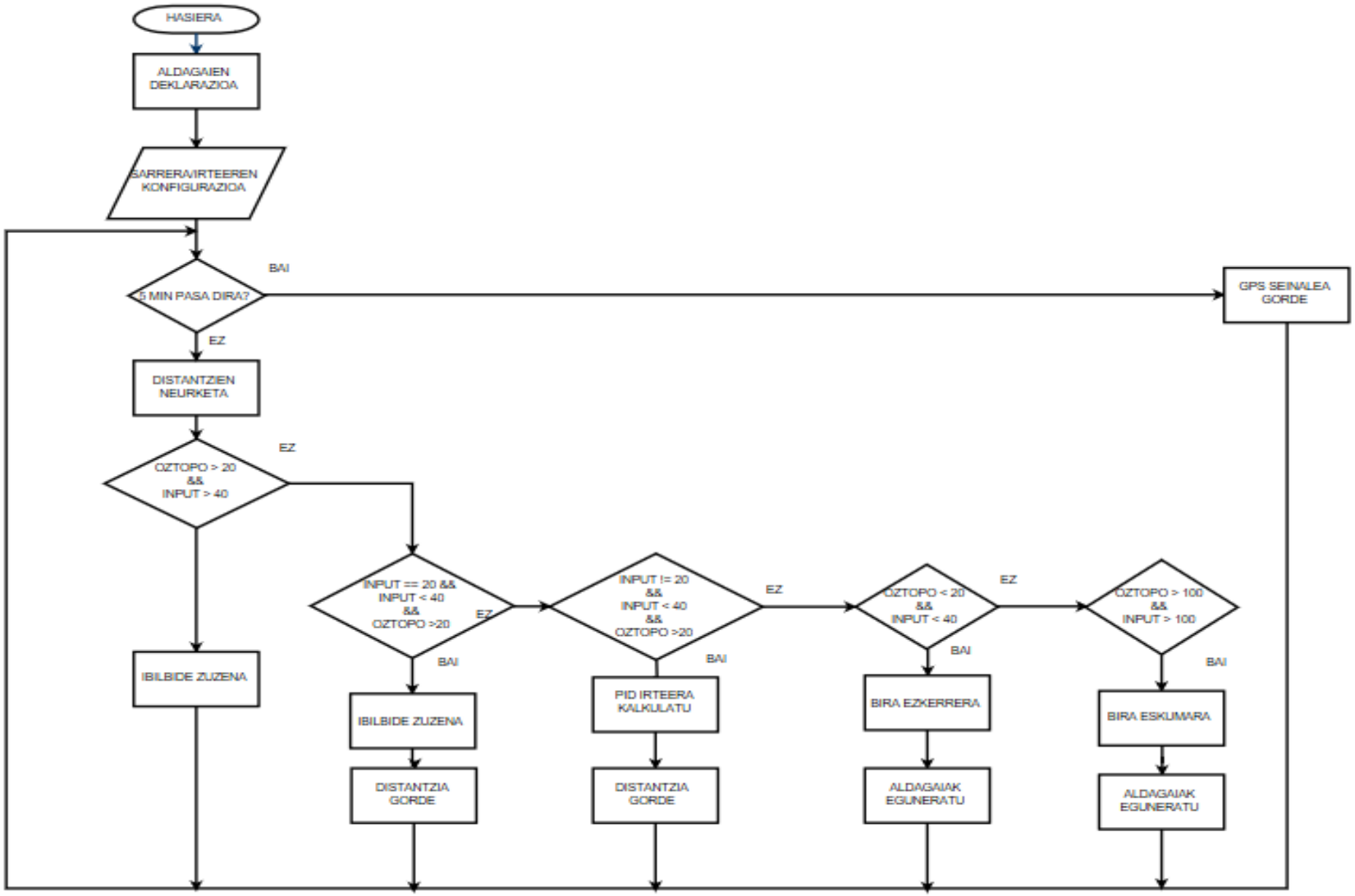
AURKIBIDEA

IRUDIEN AURKIBIDEA

TAULEN AURKIBIDEA

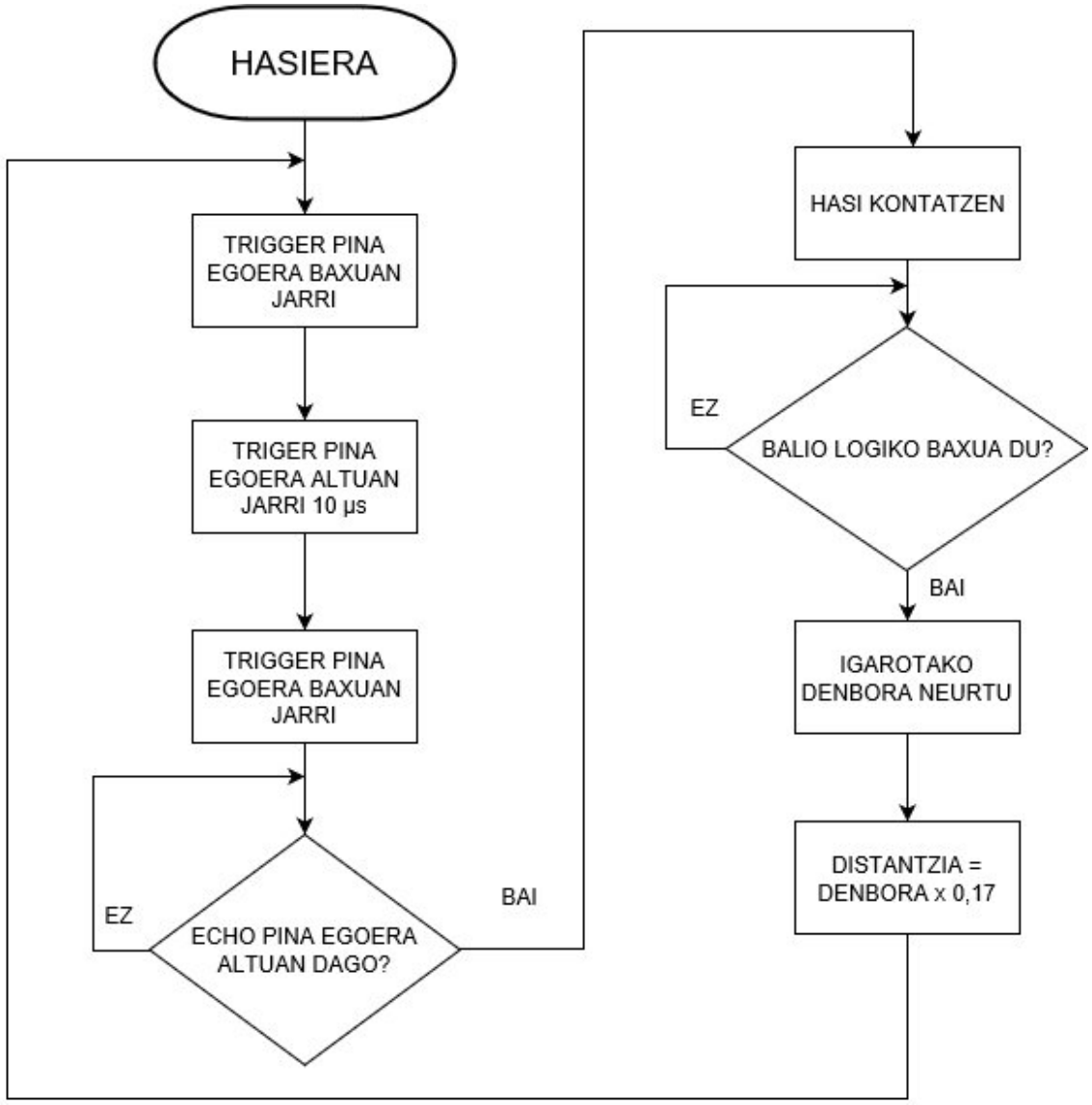
1. KODEAREN AZALPEN OROKRRA

1.1. Diagrama orokorra:



Behin autoa martxan jarrita, sentsoreen ingurunearekiko informazioa jasoko dute (5 minutu pasa badira, GPS seinaleak jasotako informazioa mikro-sd txartelean gorde beharko da). Informazioa hau erabilita egoera aztertuko da eta autoak funtzionamendu bat edo beste izango du. Objekturik detektatzen ez bada inguruan adibidez, ibilbide zuzena izango du hormaren bat edo oztopo bat topatu arte.

1.2. Distantziaren neurketa:



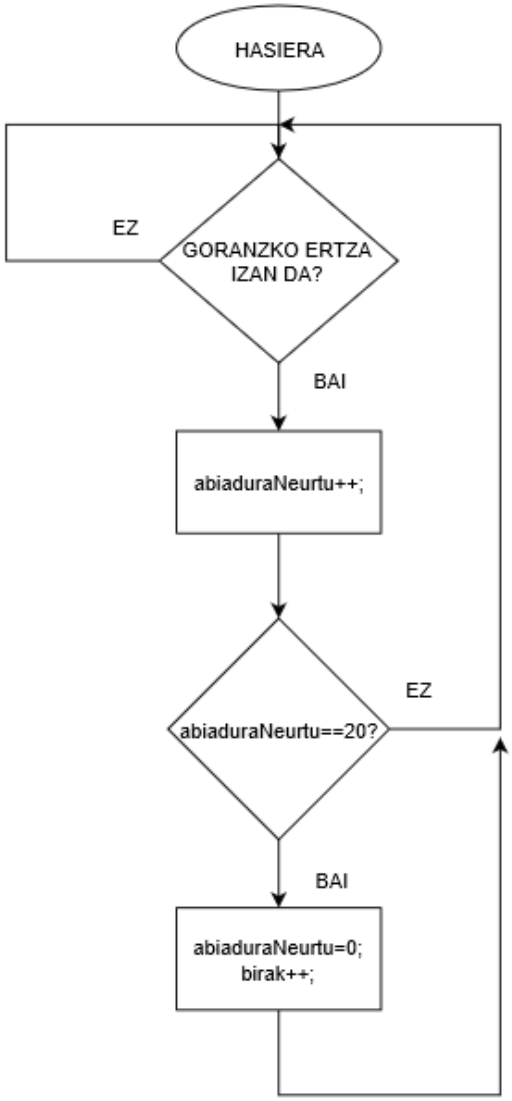
Tren pultsua bidali baino lehen, egonkortasuna emateko sentsoreari “trigger” egoera baxuan jarri beharko da. Behin 5 µs itxaronda, sentsoreak “trigger” pina egoera altuan jarri ahal izango du tren pultsu bidaltzeko. Egoera altuko balio hau 10 µs-tan mantendu beharko da neurketa zuzena izan dadin.

Behin tren pultsua bidalita, “echo” pinak oihartzunaren anplitudea neurtu beharko du. Horretarako pulseIn() funtzioa erabiliko da. Funtzio honek bi parametro izango ditu,

neurketa egiteko erabiliko den pin zenbakia eta neurtu beharreko balio logikoa. Behin ultrasoinuan joan-etorria egiteko behar izan duen denbora jakinda eta soinuaren abiadura baldintza normaletan ezaguna denez, distantzia kalkulatu ahal izango da.

1.3. Aurreratutako distantziaren neurketa:

Kodegailuak, tren pultsu bat sortuko du diskoaren eta sentsore optikoaren mugimenduaren eraginez. Irteerako seinale hau mikro-kontrolagailura bidaliko da etendurak sortzeko seinale honen goranzko ertz bakoitzean. Honi esker, sentsoreak sortzen dituen pultsu kopurua zenbatu ahal izango da eta ,ondorioz, gurpilak ematen dituen bira kopurua zenbatu ahal izango dira. Gurpilen bira kopurua eta honen erradioa ezaguna izanda, egindako distantzia lineala kalkulatu ahal izango da.

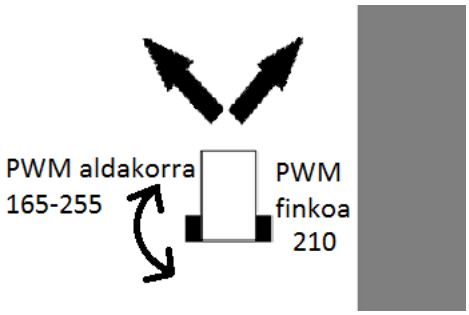


2. KONTROLA

Mikro-kontrolagailuaren eta sentsoreek eskainitako informazioaren laguntzaz motorren abiadura kontrolatu ahal izango da. Prozesu hau burutu ahal izateko ikuspuntu bat baino gehiago planteatu dira, baino kasu guztietan PID_v1 liburutegia erabiliko da. Honi esker, egin beharreko kalkuluak eta idatzi beharreko kode lerroak izugarri txikituko dira.

2.1. PID kontrolagailu bakarra erabilita, gurpil baten abiadura finkoa izanik:

Lehenengo aukera PID kontroladore bakarra erabiltzea izan da, hau da, motor baten irteera kontrolatzea eta bestea finkoa izatea. Honela, motor finkoaren abiadura kotxearen abiadura ezarriko du eta beste motorrak aldiz, autoak eramango duen norabidea. Autoa hormatik oso hurbil badago adibidez, eskumako gurpilaren abiadura ezkerreko gurpilarena baino handiagoa izango da. Kontrako egoera baten, gurpilen erantzuna kontrakoa izango da baita ere. Ikuspuntu honen abantailetako bat PID bakarra programatu beharko litzatekela izango litzateke. Hala ere, bi motorrak ezaugarri berekoak izanik PID-aren parametroak berberak izan beharko lirateke beraz ez du abantaila argi bat suposatzen.



1.irudia. PID bakarra erabilita.

Bigarren abantaila nagusia, eskumako gurpilak abiadura finkatuko duela da. Gurpil honen abiadura ezagutuz kotxearen abiadura zehaztu ahal izango da eta beste gurpila norabidea baino ez du zehaztuko.

Bestalde, badauka desabantaila argi bat. Bi motorren abiaduren arteko ezberdintasuna nahiko baxua izango da eta zailagoa izango da ibilbidea zuzentzea.

2.2. Bi PID kontrolagailu erabilita

Kasu honetan bi PID kontrolagailu ezberdin programatu beharko dira. Ezaugarri berdinak izango dituzte baina kontrako direkzioarekin, hau da, hauetako batek hormatik aldentzerakoan norabidea zuzendu ahal izateko ezkerreko motorrari bidalitako potentzia handitu beharko du. Beste PID-aren kasuan, gero eta urrunago egon orduan eta potentzia baxuago bidali beharko dio eskumako motorrari.

Honela, motorren funtzionamendu tartea 160 eta 255 tartean ezartzen badugu, muturreko kasu baten bi motorren arteko potentzia ezberdintasuna eta ondorioz, abiadura ezberdintasuna aurreko kasuaren bikoitza izango da. Honi esker, askoz errazagoa izango da autoaren norabidea zuzentzea.

2.3. PID kontrolagailua bakarra:

Kasu honetan lehenengo aukerarekin alderatuz, bi motorren PWM seinaleak aldakorrak izango dira baina PID bakarra erabilita. Honako hau lortzeko, motorrek PWM oinarrizko seinale bat izango dute maximoa baino baxuagoa izango dena, eta

seinale horri PID irteera gehitu edo kendu egingo zaio motorraren arabera. Honela, bi motorren potentzia eta abiadura ezberdintasuna aurreko kasuaren berdina izatea lortuko da.

Ikuspuntu honek dituen abantailak aurreko bi ikuspuntuek dituzten abantaila berak izango dira. Programazio aldetik pixka bat konplikatu daiteke baina PID bakarrarekin funtzionamendu zuzena lortuko da.

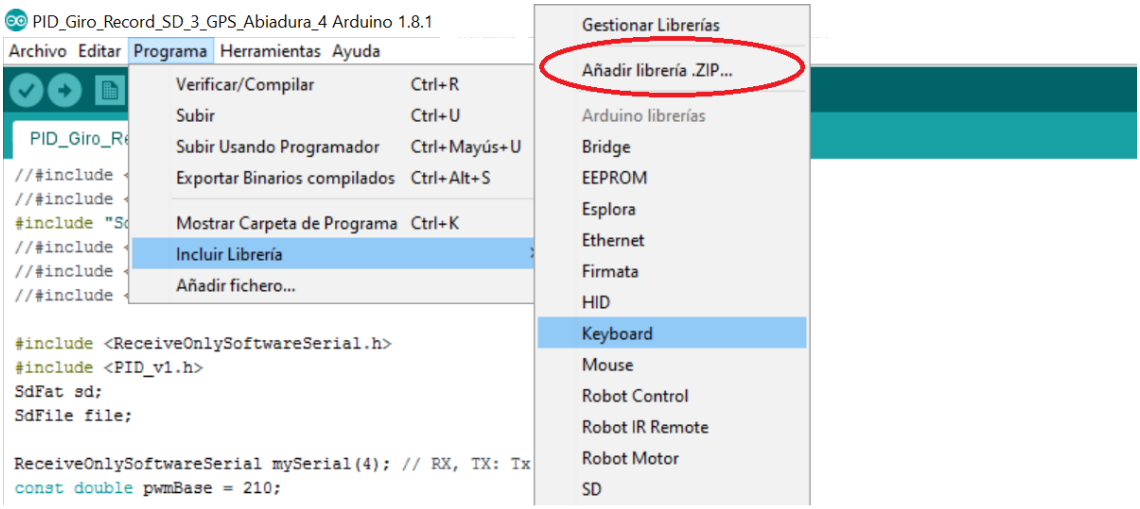
3. LIBURUTEGIAK

Mota askotako programazio proiektuetan oso lagungarria da liburutegiak erabiltzea. Askotan liburutegi hauek oso ondo dokumentatuak daude eta izugarri errazten dute erabiltzailearen lana.

Erabili beharreko liburutegia ez badago Arduino IDE softwarearen barne, pausu sinple batzuk jarraitu beharko dira behin liburutegia deskargatuta ".zip" formatuan.

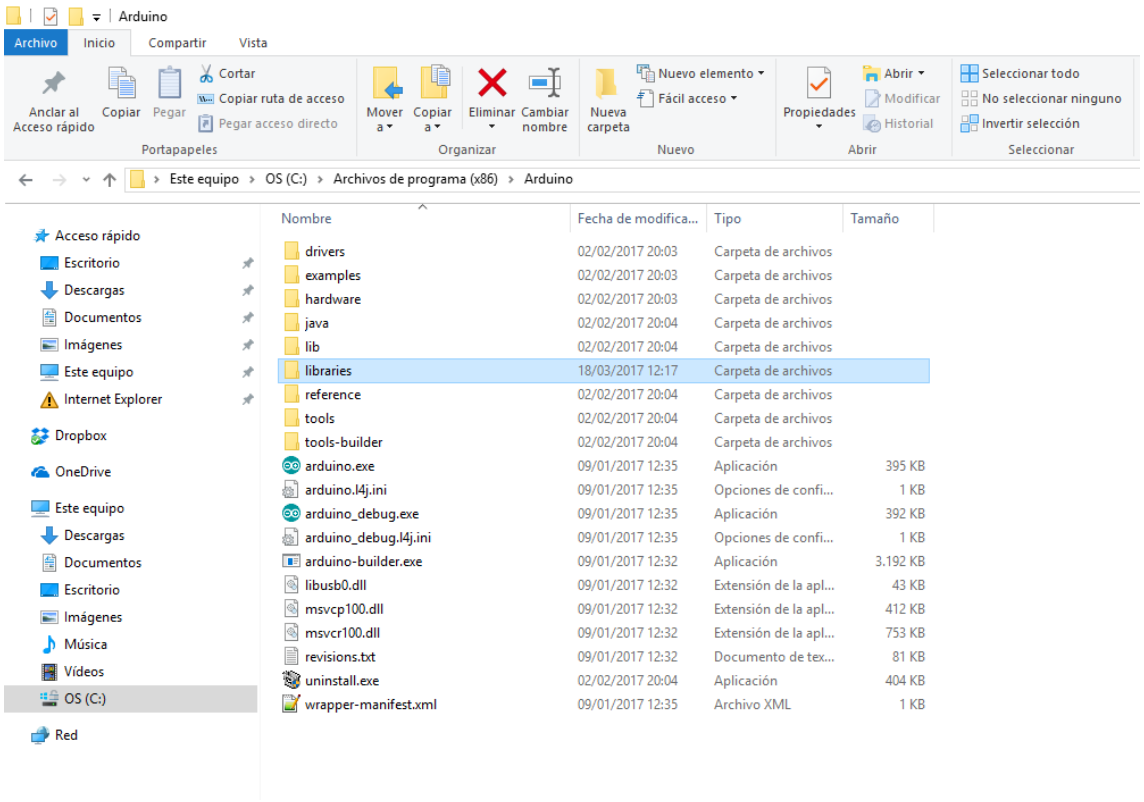
3.1. Liburutegien instalazioa

Bi modutan instalatu ahal izango dira liburutegiak, modu automatikoan softwareak eskaintzen duen "Añadir libreria" aukerarekin. Horretarako, .zip fitxategia deskargatu beharko da eta ondoren arduino softwarea exekutatuko.



2. irudia. Liburutegien instalazio automatikoa.

Behin ".zip" fitxategia aukeratuta, liburutegiaren barneko funtzioak erabiltzen hasi daiteke. Prozesu hau hala ere eskuz ahal izango da. Horretarako .zip fitxategiko fitxategiak atera eta arduinok liburutegiak gordetzen tokian itzatzi.



3.irudia. Arduino-ren liburutegiak eskuz instalatu.

Karpeta hau, irudian ikus daitekeen moduan, “C:\Program Files (x86)\Arduino” karpetan aurkitu daiteke.

3.2. Erabilitako liburutegiak

Hiru izan dira proiektuaren kodea eraikitzeko erabilitako liburutegiak:

3.2.1. ReceiveOnlySoftwareSerial

Datuen serieko transmisioa gauzatu ahal izateko mikroak UART (Universal Asinchronous Receiver Transmitter) izeneko hardware bat erabiltzen du. Hardware honi esker, ATmega prozesadoreak datuak jasotzeko gai da nahiz eta beste prozesu bitartean burutu, 64 byteko buffer seriean memoria nahikoa dagoen bitartean. Beraz, liburutegi honi esker komunikazio serie bat lortu daiteke mikro-kontrolagailuaren edozein pin digital erabilita.

115200 bps-ko transmisio abiadurak lortu daitezke eta parametro batek seinaleztapen alderantzikatua ahalbidetzen du, hau da, modu hau aktibatuz gero, LOW bat (0 volt normalean) Rx pinean bit bat adieraziko du eta HIGH bat 0-bit bat izango balitz bezala. Kontutan izan beharko da irteerako informazio seriea adierazteko balio-tartea, erabilitako mikro-kontrolagailuak jasan beharko duela, hau da, 5V-tako mikro bat erabiltzen bada beste gailuaren irteerako balio-tartea ezingo ditu 5V horiek gainditu.

Liburutegi hau software serial liburutegiaren modifikazioa baino ez da, datuak jasotzeko bakarrik (jaso eta bidali beharrean) erabiliko dena. Honi esker,

programatzerako orduan pin bakarra erabili ahal izango da eta ez bi, eta liburutegiaren objektua instantziatzerakoan ez da beharko irteerako pin bat adierazten mikroarentzat, informazioa jaso baino ez du egingo.

3.2.2. SdFat

Liburutegi honi esker, idazketak eta irakurketak egin daitezke FAT16/FAT32 motako fitxategi sistemetan, SD/SDHC motako memoriak erabiliz.

Liburutegi honek Arduino IDE softwarearen 1.6.x (edo berriagoak) bertsioaren beharra du funtzionatzeko, kasu honetan 1.6.9 bertsioa erabili da beraz ez da inolako arazorik egon.

SDconfig.h fitxategia erabili ahal izango da liburutegia konfiguratzeko. Fitxategia SdFat karpetaren barruan dagoen src karpetan aurkitu ahal izango da behin liburutegia deskargatuta izanik.

Fitxategietan izen luzeak erabiltzeko adibidez USE_LONG_FILE_NAMES makroari 0-ren ezberdina den balio bat eman beharko zaio. Hala ere, fitxategi hauen izenak 255 karaktereetara egongo dira mugatuak eta hurrengo karaktereak ezingo dira erabili:

- < (txikiago baino)
- > (handiago baino)
- : (bi puntu)
- “ (komatxoak)
- / (barra)
- \ (kontrabarra)
- | (barra bertikala)
- ? (galdera ikurra)
- * (izartxo)

Izen luzeak duten fitxategiak zabaltzeko beharrezko luzeagoa izan daiteke, hala ere, idazketen eta irakurketen errendimendua ez da aldatzen fitxategiaren izenaren motaren arabera.

SD_SPI_CONFIGURATION makroa erabiliz SPI-rekin erlazionatutako konfigurazio batzuk aktibatu daitezke, SdFatSoftSpi eta SdFatLibSpi klaseak hain zuzen. SdFatLibSpi klasea arduinoren ohiko SPI liburutegia erabiltzen du, SdFatSoftSpi software SPI erabiltzen duen bitartean.
SD_SPI_CONFIGURATION 0 baldin bada, SdFat klasea definitzen da bakarrik eta eskuragarri egonez gero, SPI-ren inplementazio azkar bat erabiltzen da.
SD_HAS_CUSTOM_SPI zero baldin bada, ordea, SPI-ren liburutegi estandarra erabiltzen da.

SD txartelaren CRC (Cyclic Redundancy Check, erroreak detektatzeko kode bat baino ez da, ustekabeko aldaketak detektatzeko datuetan) frogapena egiteko, USE_SD_CRC makroari zero ez den beste balio bat eman beharko zaio.

FAT12_SUPPORT makroa erabiliz, FAT12 formatua duten fitxategi sistemak erabiltzeko aukera izango da. Hala ere formatu hau ez dago guztiz frogatuta eta flash memoria gehigarria behar du.

3.2.3. PID_V1

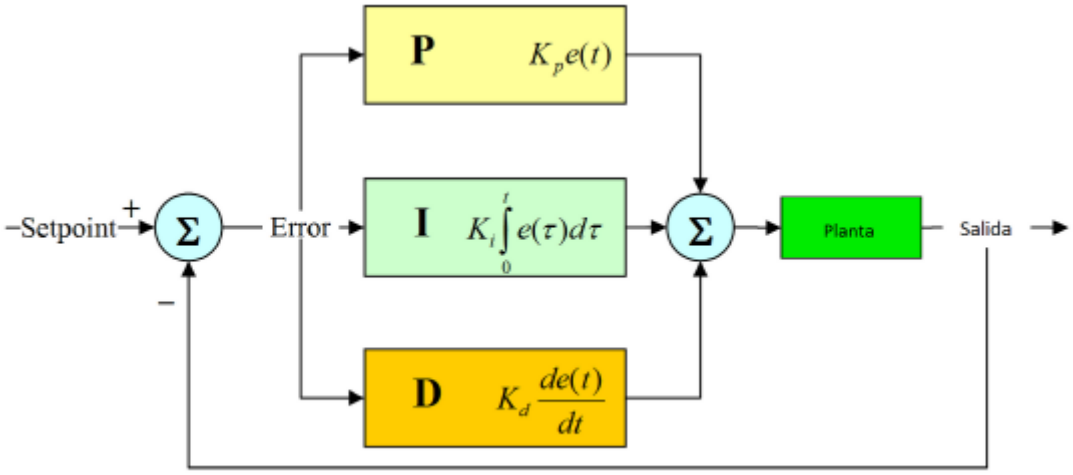
Liburutegi honi esker, PID kontrolagailuaren doiketa erabat errazagoa izango da, izan ere, PID kontrolagailu batek behar duen kode osoaren inplementazioa izugarri errazten du. Hala ere, kodearekin hasi baino lehen sistemaren seinale eta parametro ezberdinak identifikatu eta zehaztu beharko dira.

PID kontrolagailuaren ekuazioa hurrengoa izango litzateke:

$$u(t) = k_p e(t) + \frac{k_p}{T_i}\int_0^t e(t)dt + k_p T_d \frac{de(t)}{dt}$$

1.ekuazioa

Ekuazioaren osagai guztiak diagrama baten ikusita:



4.irudia. PID kontrolagailu baten diagrama. Iturria: https://ca.wikipedia.org/wiki/Proporcional_integral_derivatiu

- e(t) errore seinalea izango da, hau da, hormarekiko distantziaren eta setpoint-aren (20 cm-takoa) arteko aldea.
- u(t) kontroladorearen irteera izango da, limite batzuen artean finkatuta egongo dena.
- Kp irabazi proportzionala izango da.
- Ki integrazio-konstantea.
- Kd deribazio-konstantea.

Lehenengo kontrol blokea, konstante proportzionalari dagokiona, konstante proportzionalaren eta errore seinalearen arteko biderkadura izango da. Honekin, egoera egonkorreko errorea ia guztiz desagerraraztea lortzen da.

Kontrol integralari dagokion bigarren blokea, kontrol proportzionalak sortutako egoera egonkorreko errorea guztiz ezabatzea du helburu.

Kontrol deribatiboa ekintza proportzionalaren etorkizuneko errorea maneiatzen du. Hiru termino hauen irteera batuz, hau da, proportzionala, integrala eta diferentziala, PID kontrolagailuaren irteera kalkulatzen da. Horrenbestez, $u(t)$ delakoa kontrolagailuaren irteera bezala definitzen da. Irteerako seinale honek eragin zuzena izango du motorretara helduko den potentziarekin.

Beraz ezaugarri hauek kontutan izanda, oinarrizko kode bat garatu daiteke mikro-kontrolagailuarentzat. Hala ere, aipatu beharra dago kode hau hurbilketa bat besterik ez dela izango, mikro-kontrolagailuek denbora diskretuan egiten baitute lan.

```
/*PID kontroladorean erabilitako aldagaiak*/
unsigned long lastTime;
double Input, Output, Setpoint;
double errSum, lastErr;
double kp, ki, kd;

void Compute()
{
   /*Azken kalkulutik igarotako denbora*/
   unsigned long now = millis();
   double timeChange = (double)(now - lastTime);

   /*Errore aldagai guztiak kalkulatu*/
   double error = Setpoint - Input;
   errSum += (error * timeChange);
   double dErr = (error - lastErr) / timeChange;
  /*PID-aren irteera seinalea kalkulatu*/


  Output = kp * error + ki * errSum + kd * dErr;

  /*Aldagaien balioa gorde hurrengo iteraziorako*/
   lastErr = error;
   lastTime = now;
}

/*Sintonizazio konstanteen balioa finkatu*/
void SetTunings(double Kp, double Ki, double Kd)
{
   kp = Kp;
   ki = Ki;
   kd = Kd;
}
```

Aurreko kodean ikusten den moduan, liburutegiak planteatutako hasierako kode honetan, ez da laginketa denbora bat zehazten eta kontrol seinalearen eguneraketa tarte irregularretan egingo da. Honek arazo batzuk ekarriko ditu:

Kontrolagailuaren funtzionamendua ez da oso egonkorra izango, irregulartasunarekin exekutatuko delako.

Gehiegizko operazio matematikoak egin behar dira, izan ere, osagai deribatiboa eta integrala denboraren menpe baitaude, eta honek kalkulu horiek egin behar izatea ekarriko du.

Arazo hau eta beste batzuk konpondu ahal izateko, "Brett Beauregard"-ek liburutegiaren sortzaileak konponbide batzuk proposatzen ditu liburutegiaren funtzionamendu orokorra egonkorragoa izan dadin. Aldaketarik garrantzitsuena PID-aren irteera kalkulatzen duen funtzioa tarte erregularretan exekutatzea izango litzateke. Honi esker, osagai integralari eta deribatuari dagozkion kalkuluak asko sinplifikatzen dira. Gainera, laginketa denbora exekuzio denboran aldatzea ahalbidetzen da.

"Derivative Kick" efektua ezabatzeko, osagai deribatiboa moldatzen da hurrengo hurbilketak aplikatuz:

$$\frac{dErrorea}{dt} = \frac{dSetpoint}{dt} - \frac{dSarrera}{dt}$$

2.ekuazioa

Setpoint edo erreferentzia konstantea denean, honen deribatua 0 izango da eta hurrengo berdinaketa aplikatu ahal izango da:

$$\frac{dErrorea}{dt} = -\frac{dInput}{dt}$$

3.ekuazioa

Aldaketa honi esker, erreferentzia seinalea bat batean aldatzen denean irteeran ez dira ikusiko orain arte ikusitako perturbazio bortitz horiek.

Aldaketak egiterakoan sintonizazio parametroetan, irteeran perturbazio batzuk ikus zitezkeen baita ere, behin K_I parametroa aldatuta erroreak izandako balio guztiengatik biderkatzen baitzen. Errore hau konpontzeko, konstante integrala errorearen integralaren barne sartzea proposatzen da.

$$\int K_I\, e\, dt \;\approx\; K_I\, e_n + K_{I_{n-1}}\, e_{n-1} + \cdots$$

4. ekuazioa

Honela, aurreko hurbilketa aplikatuz, osagai integral osoa errore seinalearen eta behin-behineko konstante integralaren arteko biderkaduraren batukari bategatik ordezkatzen da, ITerm deiturikoa hain zuzen.

"Reset Windup" efektua ezabatzeko, termino integralaren (ITerm kasu honetan) eta kontroladorearen irteera mugatzen da. Honela, errorea oso handia denean (sistemak habiaratzen direnean adibidez) irteera kontrolatu bat lortuko da eta nahiz eta balio maximoa izan, errorea jaisterakoan ez da atzerapen efektu hori sortuko.

Erabiltzailea kontroladorea amatatzea nahi izanez gero, bi modu inplementatzen dira, "Automatic" eta "Manual". Honi esker, kontrolagailua

amatatu ahal izango da eta irteerari balio bat eskuz eman. “Compute” metodoa exekutatu ahal izateko modu automatikoan egon beharko da funtzionatzen kontrolagailua, eta funtzionamendu modua aukeratzeko, “SetMode” funtzioa eraldatu da “Initialize” deituriko funtzioari deia eginez. “Initialize” funtzio honetan “lastInput” bezalako aldagaiak eguneratuko dira eta ITerm aldagaiaren balioa ezarritako mugen artean finkatuko da, berriz ere funtzionamendu automatikoa erabiltzerakoan perturbaziorik ez egoteko irteeran.

PID liburutegiari eman beharreko beste parametro direkzioa izango litzateke. Parametro hau ulertu ahal izateko bi egoera planteatuko dira. Lehenengoan adibidez, labe baten kontrola gauzatu behar da eta ezarritako tenperatura erreferentzia 500 °C izanik, eta tenperatura erreala 300 °C badira, 200 °C-tako errorea izango da. Kasu honetan errorea gero eta handiagoa izan, kontroladorearen irteera orduan eta handiagoa izan beharko da tenperatura igotzen saiatzeko.

Kartografiatzeko autoan adibidez, eskumako motorearen kasuan (hormarekiko hurbilen dagoena), hormatik aldentzerakoan behin 20 cm-tako distantzia hori gaindituta errorea gero eta handiagoa izango da labearen kasuan bezala, baina kontroladorean eman beharreko irteera gero eta txikiagoa izan beharko da. Kontrako motorearekin alderantzizko funtzionamendu bat izango duenez, norabidea zuzendu ahal izango da.

Beraz, PID kontroladoreak bi egoera ezberdinei egin beharko die aurre eta parametro honen bidez adieraziko da. Alderantzizko prozesu bati aurreko egiteko PID-ak Kp, Ki eta Kd-ri zeinua aldatu beharko die funtzionamendu zuzena izateko. Horretarako, lehenik konstante hauek positiboa izatea eskatuko da, eta beste funtzio baten, “SetControllerDirection” zeinua aldatuko zaie prozesu motaren arabera.

Aldaketa guztiak aplikatu ondoren, kontrolagailuaren funtzionamendu askoz egonkorrago bat lortzen da. Liburutegiaren kodea honela geratuko litzateke:

```
unsigned long lastTime;
double Input, Output, Setpoint;
double ITerm, lastInput;
double kp, ki, kd;
int SampleTime = 1000;  //Laginketa denbora segundu bat
double outMin, outMax;
bool inAuto = false;

#define MANUAL 0
#define AUTOMATIC 1
#define DIRECT 0
#define REVERSE 1
int controllerDirection = DIRECT;


void Compute()
{
   if(!inAuto) return;
   unsigned long now = millis();
   int timeChange = (now - lastTime);
   if(timeChange>=SampleTime)
   {
      //Errore guztiak kalkulatzen dira
      double error = Setpoint - Input;
      ITerm+= (ki * error);
      if(ITerm> outMax) ITerm= outMax;
      else if(ITerm< outMin) ITerm= outMin;
      double dInput = (Input - lastInput);

      //PID-aren irteera kalkulatzen da
      Output = kp * error + ITerm- kd * dInput;
      if(Output > outMax) Output = outMax;
      else if(Output < outMin) Output = outMin;
```

```
      lastInput = Input;
      lastTime = now;
   }
}

void SetTunings(double Kp, double Ki, double Kd)
{
   if (Kp<0 || Ki<0|| Kd<0) return;

  double SampleTimeInSec = ((double)SampleTime)/1000;
   kp = Kp;
   ki = Ki * SampleTimeInSec;
   kd = Kd / SampleTimeInSec;

  if(controllerDirection ==REVERSE)
   {
      kp = (0 - kp);
      ki = (0 - ki);
      kd = (0 - kd);
   }
}

void SetSampleTime(int NewSampleTime)
{
   if (NewSampleTime > 0)
   {
      double ratio  = (double)NewSampleTime / (double)SampleTime;
      ki *= ratio;
      kd /= ratio;
      SampleTime = (unsigned long)NewSampleTime;
   }
}

void SetOutputLimits(double Min, double Max)
{
   if(Min > Max) return;
   outMin = Min;
   outMax = Max;

   if(Output > outMax) Output = outMax;
   else if(Output < outMin) Output = outMin;

   if(ITerm> outMax) ITerm= outMax;
   else if(ITerm< outMin) ITerm= outMin;
}


void SetMode(int Mode)
{
    bool newAuto = (Mode == AUTOMATIC);
```

```
    if(newAuto && !inAuto)
    {   // Funtzionamendu modua aldatuz gero, aldagaiak eguneratu
        Initialize();
    }
    inAuto = newAuto;
}

void Initialize()
{
   lastInput = Input;
   ITerm = Output;
   if(ITerm> outMax) ITerm= outMax;
   else if(ITerm< outMin) ITerm= outMin;
}

void SetControllerDirection(int Direction)
{
   controllerDirection = Direction;
}
```

4. GARATUTAKO KODEA

Atal honetan garatutako kodea eta funtzio guztien ezaugarriak eta deklaratutako aldagaiak aztertuko dira.

```
#include "SdFat.h"
#include <ReceiveOnlySoftwareSerial.h>
#include <PID_v1.h>

SdFat sd;
SdFile file;

ReceiveOnlySoftwareSerial mySerial(4); // RX, TX: Tx pina ez dugu behar.
const double pwmBase = 210;
const int arrayTamaina = 4;

#define pinOztDistTrig 8
#define pinOztDistEcho A2
#define pinAlbokoDistEcho 3
#define pinAlbokoDistTrig 6

//Iparra, mendebaldea, hegoaldea, ekialdea adieraziko dute aldagai bakoitzak.
//String dataString;
float ibilbidea[arrayTamaina] = {0,0,0,0}, iparra = 0, mendebaldea = 0,
hegoaldea = 0, ekialdea = 0;

// mugitzen = 0, record = 0,
int  gpsInterval = 0, select = 0, birak = 0;
bool hormaAurkitua = false;

//ISR barnean erabiltzen diren aldagaiak.
volatile long echo = 0;
volatile long echo_start = 0;
volatile int abiaduraNeurtu = 0;
//Volatile azalduta hemen: https://www.arduino.cc/en/Reference/AttachInterrupt
```

Kodearen lehenengo atal honetan erabilitako liburutegien eta aldagai guztien deklarazioa ikus daiteke. sd objektuari esker, SdFat eskaintzen dituen funtzioak erabili ahal izango dira, sd.Begin(9600) adibidez, “Begin” sd liburutegiaren funtzio bat izanik. “file” berriz, fitxategian idazketak egitea ahalbidetuko du.

mySerial objektua, ReceiveOnlySoftwareSerial liburutegiaren funtzioak inplementatu ahal izateko erabiliko da, GPS moduluak lortutako informazioa jaso ahal izateko eta 5 minututako tartea igaro ondoren fitxategian idatzi ahal izateko. Aurreko ataletan azaldu den moduan, pin bakarra erabiliko da liburutegi honen objektu hau instantziatzeko, izan ere, informazioa jaso baino ez denez egingo, informazioa bidaltzeko pina ez da erabiliko.

Ondoren, definitutako aldagai guztiak ikus daitezke. “pwmBase” oinarrizko abiadura izango da. Autoa hormarekiko 20 cm-tara badago, kontroladore ez da aktibatuko eta bi motorretara PWM seinale hau helduko da. Nahiz eta abiadura

maximotik nahiko urrun egon, autoak gutxitan izango du abiadura hau, izan ere, kontroladoreak periodikoki ibilbidea zuzenduko baitu laginketa-maiztasunaren arabera.

Distantziak kalkulatzeko 2 sentsore ezberdin erabili dira, baina kodearen funtzio bera. Ondorioz, sentsore biak erabili ahal izateko, erabilitako pinak parametro modura pasatzen dira neurketa funtziora. Horregatik, pinekin ez nahasteko konstanteak balira bezala deklaratu dira bakoitza bere izenarekin erabili ahal izateko funtzioei deitzerako orduan.

"ibilbidea" matrizea distantzia gordetzeko erabiliko da. Float motako lau elementu izango ditu bere barnean, horregatik "arrayTamaina" konstantea erabiltzen da matrizearen tamaina adierazteko deklaratzerako orduan. Matrize honen elementuek norabide erlatiboak (hasierako ibilbidearekiko norabidea izango baitira) adieraziko dute. Funtzionamendu normala erlojuaren aurkako zentzuan birak egitea izango denez, lehenengo elementua iparra adieraziko du, bigarrenak mendebaldea, hirugarrenak hegoaldea eta azkenak ekialdea.

```
//Define Variables we'll be connecting to
double Setpoint, Input, Oztopo, Output, doubleDistantzia, pwmEzkerra, pwmEskuma;

double kp=55, ki=5, kd=20; //Al estar en automatic ajusta automaticamente los parametros.
PID myPID1(&Input, &Output, &Setpoint, kp, ki, kd, DIRECT); //Ezkerreko motorra.

#define FILE_BASE_NAME "Data"
#define error(msg) sd.errorHalt(F(msg))
char fileName[13] = FILE_BASE_NAME "00.csv";
```

PID liburutegiarekin erlazionatutako aldagai gehienak double motakoak izango dira, hau da, float motako datuek dute zehaztasun bikoitza izango dute. Input aldagaia, alboko sentsoreak jasotako informazioa gordeko du, PID-arekin kontrolatu behar den distantzia azken finean. Output aldagaiari esker motorrei beharrezko potentzia helduko zaie norabide zuzena izateko. Output aldagaiaren balioa motorretako bati gehituko zaio eta besteari ordea, kendu. Honela, biraketa erradio txikiagoa izango da eta arinago zuzendu ahal izango da norabidea.

Kontroladorearen konstanteak eskuz sintonizatu dira, balioak ezarriz eta sistemaren funtzionamendua konprobatuz, hau da, balioak aldatu nahiko balirateke, kodea aldatu beharko litzateke eta txartelera igo garatutako kode berria. Kalkuluak arazo barik egiteko, kontroladorearen kalkuluetan erabilitako aldagaien (Input, Setpoint…) mota berekoa izan behar dira.

Hurrengo kode lerroak SD txartelean gordetako fitxategiarekin egongo dira erlazionatuta. FILE_BASE_NAME fitxategiaren izenaren oinarrizko zatia izango da. Honela, fitxategi berriak sortzen baldin badira numerazio ezberdina izango dute izenaren amaieran baina oinarria berdina izaten jarraituko du. Hurrengo definizioa idazketa egiterakoan edozein errorerik izanez gero errore mezu bat bistaratu ahal izateko erabiliko da.

Eta azkenik fitxategien izenaren amaiera ikus daiteke. Bertan agertzen diren zenbakiak aldatuz joango dira fitxategi berriak sortzen diren heinean. Gainera, irudian

antzeman daitekeen moduan, gordetako fitxategiak ".csv" formatua izango dute, datuak tauletan gordetzea erosoago izango baita ondoren irakurketak egiteko eta eskuratutako datuak maneiatzeko.

Bigarren atal honetan behin aldagai guztien deklarazioa ondoren, setup() funtzioa erabiliko da pin ezberdinen konfigurazio guztiak egiteko.

```
void setup()
{
  Serial.begin(9600);
  mySerial.begin(9600);
  pinMode(2, INPUT); //Oztopo batekiko distantzia neurtzeko(echo)
  pinMode(pinAlbokoDistEcho, INPUT); //Hormarekiko distantzia neurtzeko(echo)
  pinMode(4, INPUT); //GPS-tik datorren informazioa irakurtzeko INPUT

  pinMode(A0,OUTPUT); //Motorren kontrolerako
  pinMode(A1, OUTPUT); //Motorren kontrolerako
  pinMode(pinOztDistEcho, INPUT);//Oztopo batekiko distantzia(echo)

  pinMode(6, OUTPUT); //Hormarekiko distantzia(trigger)
  pinMode(7, OUTPUT); //SD txartelarentzako.
  pinMode(8, OUTPUT); //Oztopo batekiko distantzia (trigger)
  pinMode(9, OUTPUT);  //PWM eskumakoa.
  pinMode(10, OUTPUT); //PWM ezkerrekoa.
  digitalWrite(A0, HIGH);
  digitalWrite(A1, LOW);
```

Setup() funtzioa baten exekutatuko da bakarrik, beraz bertan hasieraketa funtzioak eta pinen funtzionamendu moduak idatziko dira adibidez.

Serial.begin(9600) funtzioa ordenagailuarekin komunikatzeko erabiliko da UART hardware baliabidearen bidez. Ez dagoenez hardware bidez beste komunikazio serie bat ezartzeko aukerarik, software bidezko komunikazio bat erabili da txartelaren eta GPS hargailuaren artean. Komunikazio horri hasiera emateko mySerial.begin(9600) funtzioa erabili da, 9600 transmisio abiadura izanik.

Distantzia sentsoreen "echo" pinetara konektatutako pinak sarrera moduan konfiguratu dira, ultrasoinuen oihartzuna detektatu beharko dutelako. Pin digital gehiago behar izan direnez, pin analogiko batzuk pin digital moduan erabili dira, A2 pina adibidez aurreko distantzia sentsorearen echo pinera konektatu da oihartzunak detektatzeko (beraz sa)4.pin digitala GPS hargailutik informazioa jasotzeko erabili da software bidezko komunikazio seriea erabiliz. Motor bakoitzaren norantza kontrolatu ahal izateko 2 pin beharko dira, hala ere, bi motorrek beti izango dutenez norantza bera pin berdinak erabili dira motor biak kontrolatzeko. Balio logiko altu edo baxu bat emanez kontrolatuko dira, beraz honako hauek irteera bezala konfiguratu dira; eta lehen esandako moduan, pin hauek pin analogikoak izango dira digital moduan erabilita.

```
//Input = distantziaKalkulatu(pinDistLateral);
Setpoint = 20;

//Ezkerreko motorraren PIDa.
myPID1.SetControllerDirection(DIRECT); //Errorea gero eta handiagoa, irteera orduan eta handiagoa.
myPID1.SetMode(AUTOMATIC); //Modu automatikoan funtzionatzeko.
myPID1.SetOutputLimits(-45, 45); //Lehen 0-45.
myPID1.SetSampleTime(100);
```

Lehen aipatutako moduan “Setpoint” aldagaia kontrolagailuak izango duen erreferentzia hormarekiko distantzia da, kasu honetan 20 zentimetrotakoa.

Hurrengo parametroei esker, kontrolagailuaren funtzionamendua definituko da. “DIRECT” modua aukeratuta errorea handitzen den heinean, irteera handituko da baita. “AUTOMATIC” moduari esker PID-ak kalkulatuko du irteera eta ez erabiltzaileak. Beste aukera “MANUAL” izango litzateke erabiltzaileak eskuz aukeratzeko PID-aren irteera, baina kasu honetan ez da erabiliko.

Hurrengo aldagaiekin irteeraren mugak ezarriko dira. Kasu honetan kontroladorearen irteera oinarrizko abiadura bati (210-eko PWM seinalea) batuko edo kenduko zaionez, 255-210 = 45 balioa aukeratu da. Balio negatiboa izatekotan irteeran, kontrako motorrari batuko zaio irteera hau eta besteari ,berriz, kendu. SampleTime funtzioarekin Compute funtzioa zenbatero exekutatuko den aukeratu daiteke, honela laginketa maiztasuna zehaztuz. Kasu honetan 100 ms aukeratu dira, hala ere, liburutegiaren .cpp fitxategian ikus daitekeen moduan balio lehenetsia da.

```
PID::PID(double* Input, double* Output, double* Setpoint,
        double Kp, double Ki, double Kd, int ControllerDirection)
{

    myOutput = Output;
    myInput = Input;
    mySetpoint = Setpoint;
    inAuto = false;

    PID::SetOutputLimits(0, 255);                //default output limit corresponds to
                                                 //the arduino pwm limits

    SampleTime = 100;                            //default Controller Sample Time is 0.1 seconds

    PID::SetControllerDirection(ControllerDirection);
    PID::SetTunings(Kp, Ki, Kd);

    lastTime = millis()-SampleTime;
}
```

5.irudia. PID liburutegiaren balio lehenetsiak.

Hurrengo atalean datuak gordetzeko fitxategiaren izena zehaztuko da. Sistema pizten denetik amatatu arte fitxategi bakarra sortuko da horregatik setup funtzioan jarri da kode atal hau.

```
const uint8_t BASE_NAME_SIZE = sizeof(FILE_BASE_NAME) - 1;
if (!sd.begin(7, SPI_HALF_SPEED)) {

  sd.initErrorHalt();
}
if (BASE_NAME_SIZE > 6) {
  error("FILE_BASE_NAME too long");
}
while (sd.exists(fileName)) {
  if (fileName[BASE_NAME_SIZE + 1] != '9')
  {
     fileName[BASE_NAME_SIZE + 1]++;
  }
  else if (fileName[BASE_NAME_SIZE] != '9')
  {
    fileName[BASE_NAME_SIZE + 1] = '0';
    fileName[BASE_NAME_SIZE]++;
  }
  else
  {
    error("Can't create file name");
  }
}
//Serial.println(fileName);
if (!file.open(fileName, O_CREAT | O_WRITE | O_EXCL)) {
  error("file.open");
}
```

Fitxategiaren izena “8.3 filename” formatua izan beharko du, hau da, fitxategiaren izenak 8 karaktere izango ditu gehienez, zenbakiak barne (horregatik “BASE_NAME_SIZE” aldagaia ezingo da 6 baino handiagoa izan). Eta fitxategi motaren luzapena, 3 karaktereekin adierazi beharko da. Kasu honetan fitxategiaren izena beti izango da “DataXX” X-ak numerazioa adieraziz eta hedapena “.csv”, beraz ez da inolako arazorik egongo.

Ondoren, SD txartelarekin SPI komunikazioa hasiko da konprobatzeko bertan dauden fitxategien izenak. Fitxategiaren numerazioa konprobatzeko, lehenengo eskumako digitua konprobatuko da 9 baino txikiagoa dela 1 gehitzeko eta beste fitxategi ezberdin bat sortzeko. Zenbaki hau 9 baldin bada, 0 jarriko da posizio honetan eta ezkerreko digituan 1 gehituko da, 1 bat lortuz orain arte 0 baitzen. Honela fitxategiaren numerazioa gehitzen joango da 99-ra heldu arte. Momentu honetan sistemak errore bat emango du izena ezin dela sortu adieraziz, zenbakia 100 izango balitz izen osoak 9 karaktere beharko zituelako. Fitxategia sortuko da eta bertan idazteko aukerarekin zabalduko da. Erroreren bat egonez gero, “file.open” mezua adieraziko da. Fitxategia dagoeneko zabalduta egongo balitz, sistemak errore bat emango luke baita ere SdFile liburutegian ikus daitekeen moduan:

```
uint8_t SdFile::open(SdFile* dirFile, const char* fileName, uint8_t oflag) {
  uint8_t dname[11];
  dir_t* p;

  // error if already open
  if (isOpen())return false;
```

setup() funtzioarekin bukatzeko “attachInterrupt” funtzioa erabiliko da kodetzaileak-ak sortuko dituen etendurak deklaratzeko. Kodetzaileak-ak pultsu tren bat sortuko du autoak aurrera egiten duen bitartean, horregatik kodea exekutatzen den bitartean etendurak sortzea erabaki da, kodetzaileak sortutako seinalearen goranzko ertz bakoitzean. Honela, ez da pultsurik galduko eta denbora errealen exekutatuko da.

```
  attachInterrupt(0,rising, RISING);
}
```

Funtzio honen lehenengo parametroa etenduraren zenbakia izango da eta bi aukera izango dira, 0 edo 1. Zenbaki hau erabilitako pinaren araberako izango da, txartelaren bigarren pin digitala erabiltzen bada sortutako etendurak 0 zenbakia izango du eta 3.pin digitala erabiltzen bada 1 zenbakia. Bi pin hauek erabili ahal izango dira bakarrik kanpotiko etendurak sortzeko. Hurrengo parametroarekin etendurak deituko duen funtzioa adieraziko du, hau da, etenduraren ISR-a (“Interrupt Service Routine”). Azkenik, etendura noiz habiaratuko den adierazten da. Lau aukera posible daude:

LOW, pina balio baxua duenean.

CHANGE, pinaren balioa aldatzen denean.

RISING, pina balio logiko baxutik balio logiko altura aldatzen denean.

FALLING, balio logiko altutik balio logiko altura pasatzerakoan.

Etendura zenbakia txartelaren araberakoa izango da hurrengo taulan ikus daitekeen moduan:

Taula 1. Etendurak txartel ezberdinetan.

TXARTELA	**Int.0**	**Int.1**	**Int.2**	**Int.3**	**Int.4**	**Int.5**
Uno, Ethernet	2	3				
Mega2560	2	3	21	20	19	18
Leonardo, Micro	3	2	0	1	7	
Due, Zero…	Etendura zenbakia = pin zenbakia					

Hirugarren atal honetan “loop” funtzioa aztertuko da. Funtzioa hau modu ziklikoan exekutatuko da. Bertan ingurunea aztertuko da eta sentsoreetatik jasotako informazioaren arabera funtzionamendu modu ezberdinak izango ditu kotxeak.

```
void loop()
{
  float aurreratutakoDistantzia = egindakoDistantzia();

  if(millis()-gpsInterval < 300000) //5 minutu-ro GPS-aren seinalea irakurri.
  {

    Input = distantziaKalkulatu(pinAlbokoDistTrig, pinAlbokoDistEcho); //pinAlbokoDist
    //Serial.print("Hormarekiko distantzia: ");
    //Serial.println(Input);
    Oztopo = distantziaKalkulatu(pinOztDistTrig, pinOztDistEcho); //pinAurrekoDist
    //Serial.print("Oztopo batekiko distantzia: ");
    //Serial.println(Oztopo);
    else if(Oztopo > 20 && Input > 40 && hormaAurkitua == false) //IBILBIDE ZUZENA hormaren bat aurkitu arte.
    {
        analogWrite(10, 255);
        analogWrite(9, 255);
        switch(select)
         {
            case 0:
              //mugitzen = millis()-record;
              iparra = aurreratutakoDistantzia;
              ibilbidea[select] = iparra;
              break;
            case 1:
              mendebaldea = aurreratutakoDistantzia;
              ibilbidea[select] = mendebaldea;
              break;
            case 2:
              hegoaldea = aurreratutakoDistantzia;
              ibilbidea[select] = hegoaldea;
              break;
            case 3:
              ekialdea = aurreratutakoDistantzia;
              ibilbidea[select] = ekialdea;
              break;
        }
    }
```

Egindako lehenengo gauza “aurreratutakoDistantzia” aldagaia eguneratzea izango da. Aurrerago ikusiko den moduan, egindakoDistantzia() funtzioak “birak” aldagaia erabiliko du egindako distantzia kalkulatzeko.

Ondoren, “gpsInterval” aldagaiari esker 5 minutu pasa diren edo ez kalkulatuko da, 5 minuturo GPS-aren informazioa jasoko baita. Distantziak kalkulatuko dira, alboko distantziarekin hasiz eta aurrekoarekin jarraituz. Balio hauen arabera kode bloke ezberdinak exekutatuko dira.

Hiru funtzionamendu ezberdin zehazten dira, lehenengoa ibilbide zuzena. Kasu honetan autoa hormarekiko 40 cm-tako baino distantzia handiago batera dago eta ez du oztoporik izango aurrean (20 cm-tara behintzat). Funtzionamendu modu honetan zuzen egingo du oztoporen bat edo horma aurkitu arte. Horma edo oztoporen bat aurkitzerakoan, kotxeak aukera ezberdinak izango ditu. Aztertutako lehenengoa ibilbide paraleloa izango da. Funtzionamendu hau izateko horma 40 cm baino distantzia txikiagotara egon beharko da eta kasu honetan PID-a izango da norabide zuzenketaren arduraduna. Hala ere, funtzionamendu honen barne kotxea 20 cm-tara zehazki egotea planteatzen da baita ere. Kasu honetan kotxeak zuzen egin beharko du ez baitu errorerik sumatuko.

```
else if(Oztopo > 20 && Input < 40) //IBILBIDE PARALELOA
{
  hormaAurkitua = true;
  if(Input == 20)
  {
     analogWrite(10, 255);
     analogWrite(9, 255);
     switch(select)
     {
        case 0:
          //mugitzen = millis()-record;
          iparra = aurreratutakoDistantzia;
          ibilbidea[select] = iparra;
          break;
       case 1:
          mendebaldea = aurreratutakoDistantzia;
          ibilbidea[select] = mendebaldea;
          break;
       case 2:
          hegoaldea = aurreratutakoDistantzia;
          ibilbidea[select] = hegoaldea;
          break;
       case 3:
          ekialdea = aurreratutakoDistantzia;
          ibilbidea[select] = ekialdea;
          break;
    }
    //Ez dugunez norabidea aldatu select aldagaia momentuz 0 izango da, hau da, iparra.
  }
```

Kode bloke hau exekutatzerakoan egindako lehengo gauza, “hormaAurkitua” aldagaiari true balioa ematea da. Honi esker, kotxeak izkinaren bat aurkitzen badu eskumarako bira egin ahal izango du arazorik gabe. Funtzionamendu modu honetan bi aukera egongo dira, lehenengoa kotxea hormatik 20 cm-tara zehazki egotea. Kasu honetan kotxea zuzen jarraitu beharko du erroreren bat detektatu arte. PID kontrolagailuak ez du ezer egingo eta kotxearen abiadura maximoa izango da. Bigarren aukera erroreren bat egotea izango da, hau da, kotxea hormatik 20 cm baino hurbilago egongo da edo 20 cm baino urrunago. Egoera honetan, kontrolagailua bere helburua bete beharko du irteeraren arabera ibilbidea zuzenduz.

Kotxeak aurrera egiten duen bitartean eta kodearen iterazioak betetzen diren bitartean, egindako distantziaren balioa gordeko da “ibilbidea” matrizean “aurreratutakoDistantzia” aldagaiari esker. “select” aldagaiarekin aukeratuko da distantzia hori matrizearen zein posiziotan gorde. Eskumarako edo ezkerrerako birak egiten badira, norabidea 90° aldatu dela suposatuko da eta “select” aldagaiari 1 gehituko edo kenduko zaio biraren arabera.

```
else if(Input != 20 && Input < 40 && Oztopo > 20) //IBILBIDE PARALELOA
{
  hormaAurkitua = true;
  myPID1.Compute();

  pwmEskuma = pwmBase + Output;
  pwmEzkerra = pwmBase - Output;

  //Serial.print("PWM eskuma: ");
  //Serial.println(pwmEskuma);
  //Serial.print("PWM ezkerra: ");
  //Serial.println(pwmEzkerra);

  analogWrite(10, pwmEzkerra);
  analogWrite(9, pwmEskuma);

  switch(select)
  {
    case 0:
        iparra = aurreratutakoDistantzia;
        ibilbidea[select] = iparra;
        break;
    case 1:
        mendebaldea = aurreratutakoDistantzia;
        ibilbidea[select] = mendebaldea;
        break;
    case 2:
        hegoaldea = aurreratutakoDistantzia;
        ibilbidea[select] = hegoaldea;
        break;
    case 3:
        ekialdea = aurreratutakoDistantzia;
        ibilbidea[select] = ekialdea;
        break;
  } //Ez dugunez norabidea aldatu select aldagaia momentuz 0 izango da, hau da, iparra.
}
```

Output aldagaia, lehenengo atalean azaldu den moduan, (-45,45) balio tarte bat izango du eta honen arabera motore bati irteera hau gehituko edo kenduko zaio. Honela, kotxeak biratzea lortuko du ibilbidea zuzenduz.

Ezkerrera bira egiterakoan edo lehen ikusitako moduan ibilbide paraleloarekin hasterakoan, “hormaAurkitua” aldagaia true balioa hartuko du, eta honi esker birak eskumara egin ahal izango ditu.

```
else if(Oztopo < 20) //BIRA EZKERRERA
{
  hormaAurkitua = true;
  select++;
  if(select>3) select = 0;

  aurreratutakoDistantzia = 0; //Birak = 0 eginez, ondoren aurreratutakoDistantzia
                               //erreseteatu beharko litzateke, hala ere,
                               //aurreratutakoDistantzia erreseteatuko dugu ere.
  birak = 0;

  digitalWrite(A0, LOW);
  digitalWrite(A1, LOW);
  int hasi = millis();
  while((millis()-hasi) < 800)
  {
    analogWrite(10, 0);
    analogWrite(9, 255);
    digitalWrite(A0, HIGH);
    digitalWrite(A1, LOW);
  }
}
```

"select" aldagaia 3 balioa baldin badu eta bira egiten badu ezkerrera, norabidea ekialdetik iparrera aldatuko da. Beraz, select aldagaia 0 balioa hartuko du berriro ere. "aurreratutakoDistatnzia" aldagaia erreseteatu beharko da izan ere, distantzia beste norabide baten egingo da eta 0-tik hasi beharko da neurtzen.

Bira egiteko, motorrak gelditu beharko dira, motor bakarra jarriko da martxan bira hasteko eta "hasi" aldagaiari esker kalkulatuko da zenbat denbora egon den biratzen. 800 ms-tan biratutakoan funtzionamendu paraleloarekin hasteko gai izango da.

Eskumarako bira pixka bat konplexuagoa izango da, distantzia batzuk errespetatu beharko direlako funtzionamendu paraleloarekin hasi ahal izateko behin bira eginda. "select" aldagaiari adibidez 1 kendu beharko zaio gehitu beharrean, izan ere, kontrako norabide aldaketa bat izango da. Norabidea iparra baldin bada adibidez, matrizearen lehenengo elementua, eta "select" aldagaiari 1 kentzen bazaio eskumarako bira bat egin duelako, honek izango duen balioa 0 baino txikiagoa izango da eta 3 balioa esleitu beharko zaio. Honela, "select" aldagaia ekialdera apuntatu ahal izango du eta idazketan behar den tokian egingo dira.

```
else if(Oztopo > 100 && Input > 100 && hormaAurkitua == true) //BIRA ESKUMARA, hormaAurkitua = false bada,
                                                              //ez da hasieratik etengabe biratzen hasiko
{
    select--;
    if(select < 0) select = 3;

    aurreratutakoDistantzia = 0;
    birak = 0;

    analogWrite(10, 255);
    analogWrite(9, 255);
    delay(1500);

    digitalWrite(A0, LOW);
    digitalWrite(A1, LOW);
    int hasi = millis();
    while((millis()-hasi) < 800)
    {
      analogWrite(10, 255); //Hoberena 4 pin digital erabiltzea izango litzateke baina horrela
                            //2 aurreztuko ditugu.
      analogWrite(9, 0);
      digitalWrite(A0, HIGH);
      digitalWrite(A1, LOW);
    }
    analogWrite(10, 255);
    analogWrite(9, 255);
    delay(1000);
}
```

Behin aldagaiak eguneratuta, kotxeak aurrera egin beharko du hormaren hurrengo aurpegitik 20 cm aldentzeko. Behin aldenduta, bira egingo du 800 ms motor bakarra aktibatuta izanik, ezkerreko motorra hain zuzen. Ondoren aurrera egingo du segundo oso bat sentsoreei kasu egin barik hormaren hurrengo aurpegira hurbiltzeko eta alboko sentsorearekin detektatu ahal izateko.

Void loop() funtzioaren azkenengo zatian idazketak egingo dira SD txartelean. Matrizearen (norabide bakoitzean egindako distantziak izango direnak) balioak banan-banan idatziko dira idazketa independenteetan. Horretarako for loop bat erabiliko da.

```
for(int i=0;i<=3;i++)
 {
   file.println(ibilbidea[i]);
   if(!file.sync() || file.getWriteError())
   {
     error("Write error");
   }
 }
 file.println("--");
 if(!file.sync() || file.getWriteError())
 {
   error("Write error");
 }
```

Behin datu guztiak idatzita, “--” ikurrak idatziko dira datuak argiago ikusteko eta norabidea ez nahasteko.

Norabideen idazketekin batera, GPS hargailuak jasotako datuen idazketa egingo da. Kode bloke hau 5 minuturo exekutatuko da. Datuak, software bidezko komunikazio serietik jasoko dira eta SD txartelean idatziko dira baita ere.

```
else //GPS-ak lortutako datuak txartelan idatzi
{
  if(mySerial.available()>0)
  {
    file.println("GPS-ak lortutako datuak hauek dira:");
    file.println(mySerial.read());
  }

  gpsInterval = millis();
}
```

Azkenik "gpsInterval" aldagaia eguneratuko da beste 5 minutu itxaron ahal izateko hurrengo idazketa egin arte. Honekin loop() funtzioa bukatuko da.

Aurkituko diren hurrengo funtzioak, funtzio laguntzaileak izango dira distantziak kalkulatzeko adibidez.

```
double distantziaKalkulatu(int triggerPin, int echoPin)
{
  long distantzia, denbora;

  digitalWrite(triggerPin, LOW); //Distantzia sentsorearentzakodigitalWrite(9,LOW);
  delayMicroseconds(5);
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(10); //10ms egon behar da gutxienez egoera logiko altuan
  digitalWrite(triggerPin, LOW);
  denbora = pulseIn(echoPin, HIGH);
  distantzia = 0.017*denbora; //echo
  doubleDistantzia = (double)distantzia;
  return doubleDistantzia;
}
```

"distantziaKalkulatu" funtzioak bi parametro izango ditu, int motako bi aldagai distantzia neurtzeko erabiliko diren pinak adieraziko dituztenak. Funtzioak double motako balio bat itzuliko du distantziaren balioarekin, horretarako sentsoreak pultsu tren bat bidaliko du trigger pinari balio logiko altu bat eman ondoren eta pulseIn funtzioari esker ultrasoinu pultsu horren oihartzunak iraun duena kalkulatuko da. Soinuaren abiadura 343 m/s direnez baldintza normaletan, cm-pasatuz gero 34300 cm/s izango dira eta soinua joan etorria egin beharko duenez erdia neurtu beharko da, beraz zati bi. Horregatik denbora aldagaia bider 0,017 konstanteagatik bidertzen da. Bukatzeko long formatutik double formatura pasatzen da datua itzultzeko.

```
void rising(){
  //attachInterrupt(0, falling, FALLING);
  //if(abiaduraNeurtu == 0)
  //{
  //  echo_start = micros();
  //}
  abiaduraNeurtu++;
  if(abiaduraNeurtu == 20)
  {
    attachInterrupt(0, falling, FALLING);
  }
}
void falling(){
  attachInterrupt(0, rising, RISING);
  //echo = micros() - echo_start;
  abiaduraNeurtu = 0;
  birak++;
}
```

Bi funtzio hauekin autoaren gurpilek emandako bira kopurua kalkulatu ahal izango da.

Kodegailuaren irteera txartelaren 2.pin digitalera konektatuko da kanpotiko etendurak sortzeko kodegailuak sortutako seinalearen goranzko ertz bakoitzean. 20 ertz zenbatu ondoren, “falling” funtzioa deituko da eta birak aldagaiari bat gehituko zaio “abiaduraNeurtu” aldagaia erreseteatu ondoren.

“Birak” aldagaia norabidea aldatzerakoan erreseteatuko da bakarrik. Orain arte erabilitako “echo” eta “echo_start” aldagaiak ez dira erabiliko, ez baita zenbatuko 20 pultsu horiek betetzeko behar izan duen denbora ez delako abiadura neurtuko. Soilik egindako distantzia kalkulatuko da.

loop() funtzioaren iterazio bakoitzean egindakoDistantzia() deituko da “aurreratutakoDistantzia” eguneratzeko. Gurpilaren erradioa 0.0325 m dituenez, hurrengo formula erabilita egindako distantzia lineala kalkulatu ahal izango da, “P” gurpilaren perimetroa izanik eta r erradioa:

$$P = 2\pi r$$

5.go ekuazioa

```
float egindakoDistantzia()
{
 float distantzia = 0;
 if(birak != 0)
 {
  distantzia = birak * 2 * 3.14159 * 0.0325;
  return distantzia;
 }
 else
 {
  return 0;
  }
}
```

5. BIBLIOGRAFIA

PID liburutegiaren funtzioen azalpena:

http://playground.arduino.cc/Code/PIDLibrary

PID liburutegiaren erabilpen gida:

http://playground.arduino.cc/Code/PIDLibrary

AttachInterrupt:

https://www.arduino.cc/en/Reference/AttachInterrupt

ReceiveSoftwareSerial:

https://www.arduino.cc/en/Reference/SoftwareSerial

GPS modulua datasheet:

https://www.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_%28GPS.G6-HW-09005%29.pdf?utm_source=en%2Fimages%2Fdownloads%2FProduct_Docs%2FNEO-6_DataSheet_%28GPS.G6-HW-09005%29.pdf