***Sistem de parcare***

Requirements

1. Senzorii trebuie să detecteze cu exactitate limita de parcare. Cei 5 senzori vor detecta pe o lungime de 150 cm, poziționați pe bara din spate, la 15° fiecare, la .... cm distanță unul de celălalt. Conurile de detecție nu se intersectează. Toată logica de control trebuie realizată de Arduino. Datele primite de către Arduino sunt trimise mai departe prin bluetooth către Raspberry. Aici prelucrăm datele și realizăm interfața grafică.

2. Senzorii vor oferi o perspectivă clară asupra locației mașinii în raport cu obiectul. Trebuie să emită o avertizare sonoră (cu ajutorul unor buzzere) cu 15 cm înainte de obstacol.

3. Nu trebuie să aibă sensibilitate ridicată (în cazul în care sunetul este absorbit de materiale, în cazul în care apar perturbații mecanice asupra lui).

4. Raspberry-ul trebuie configurat pentru a primi date de la Arduino prin bluetooth.

5. Interfața grafică va fi creată în QT.

6. Dacă e trasă frâna de mână, avertizarea sonoră nu este activă.

***-Hardware***

Sistemul de parcare integrează următoarele componente:

* Plăcuță Arduino
* 5 senzori ultasonici (pentru măsurarea distanței)
* Un modul Bluetooth(pentru realizarea comunicației și coordonării între stemul de parcare și un computer Raspberry PI, cu rol de afișare)
* Un Buzzer cu rol de avertizare acustică
* O frână de mână simulată prin intermediul unui switch
* Fire pentru conectarea componentelor

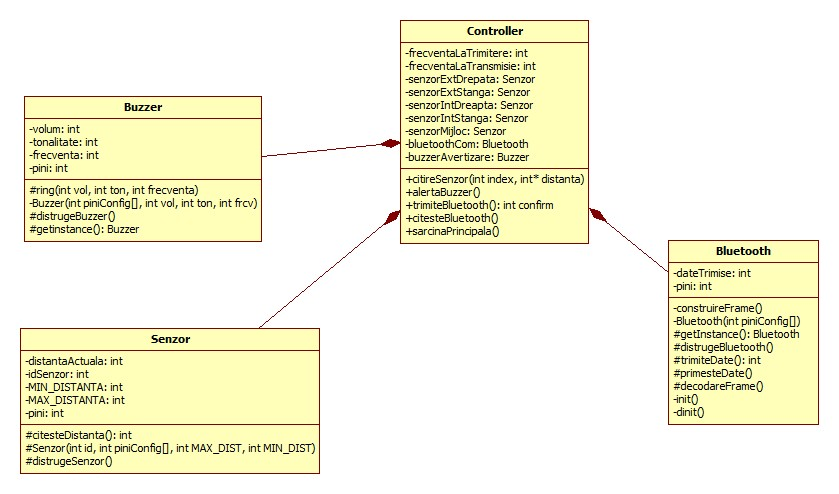
Din punct de vedere hardware, sistemul oferă următoarele facilități: asistentă la parcare, operarea pe o distanță de maxim 1,50 metri, avertizare sonoră în funcție de distanța până la cel mai apropiat obstacol și asistență grafică în timpul efectuării parcării.

Asistența grafică este oferită prin intermediul unui modul Raspberry PI care oferă, prin intermediul unui ecran o imagine de ansamblu asupra locului în care se va efectua parcarea.

Datorită necesității existenței unei comunicări între interfața utilizator și sitemul integrat care de parcare, este necesar schimbul de date între cele 2 plăcțeȘ Arduino și Raspberry PI. Aceasta se va realiza prin intermediul modului Bluetooth.

***-Software:***

Designul sistemului de parcare, din punct de vedere al structurii codului este reprezentată în diagrama de clasă de mai jos:



!Adițional fișierelor de clase, programul va conține 2 fișiere adiționale: unul pentru definirea Tipurilor de date folosite în cadrul sistemului și unul pentru definirea configurărilor de sistem.

Se va utiliza o metodă de programare modulara.

Scopul principal al claselor din diagrama de mai sus este definirea clara funcționalităților minime care trebuie oferite de program, prin intermediul componentelor hardware disponibile la nivelul acestuia.

Modulul principal, care asigură coordonarea și interacțiunea dintre cele 3 module (Bluetooth, Senzor și Buzzer) este Controller-ul. Acesta instanțiază obiecte Singleton (Buzzer și Bluetooth), precum și obiecte de tip Senzor, dar și variabile pentru controlarea comportamentului sistemului, în funcție de cerințele clientului. Principala metodă a acestei clase este operatiePrincipala() care asigură funcționarea continuă a sistemului prin citirea și prelucrarea informațiilor de la senzori, și trimiterea și primirea de informații de la modulul Raspberry.

Funcții auxiliare care vor ajuta la realizarea conexiunnilor cu celelalte clase se vor ocupa de sarcini ca: citirea senzorilor(„citireSenzor(int index, int \*dist)”-această funcție transmite printr-un pointer valoarea distante citit de la un senzor, în funcție de indexul acestuia), monitorizarea alarmei buzzer-ului („alertăBuzzer()”), trimitrea și citirea informațiilor prin intermediul modului Bluetooth(„trimiteBluetooth()” - această funcție furnizeazp și un mesaj de confirmare sau infirmare a succesului procedeului de transmisie-, „citesteBluetooth()”).

Modulele Bluetooth, Senzor și Buzzer vor fii detaliate mai jos:

**Buzzer**

Funcția pe care o îndeplinește dispozitivul numit buzer in proiect este acea de a atenționa șoferul prin intermediul sunetului, în funcție de distanta rămasa față de obiectul din imediata vecinătate in raport cu vehiculul.

Sunetul emis de buzzer va trebui sa varieze in funcție de distanta pe care vehiculul o are in raport cu un posibil obstacol. Parametri prin care va varia sunetul emis de buzzer sunt: frecventa, durata si amplitudinea (se poate implementa această utilitate cu ajutorul unui potențiometru ,daca nu este posibilă ajustarea software).

Sunetul emis de buzzer nu va fi unul deranjant sau stresant pentru a nu distrage atenția șoferului de la parcare. Acesta va înceta daca acționa frâna de mana; aceasta funcție este necesara pentru a nu deranja ocupanții vehiculului pe durata de timp in care vehiculul este staționar si pericolul ca mașina sa se deplaseze este mica.

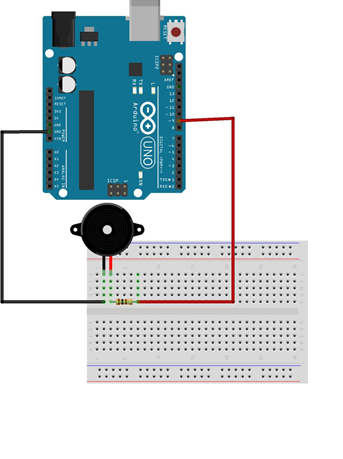
Mai departe vom prezenta elementele hardware si software folosite in implementarea funcțiilor si cerințelor prezentate mai sus :

**Hardware :**

Din punctul de vedere al hardware – ului se va folosi dispozitivul numit „**Piezo buzzer” ,** pentru a asigura generarea sunetului. Aceste dispozitive sunt folosite deoarece au greutatea redusă, construcția simplă și prețul redus .



Este recomandat ca o rezistenta de 100 ohmi sa fie conectata in serie ,pentru a asigura protecție împotriva schimbărilor bruște in curent.



**Software :**

Software – ul se va implementa folosind limbajul de programare C/C++ si, pentru compilare, se poate folosi „Arduino IDE”. Toate funcționalitățile atribuite sistemului de notificare sonora vor fi scrise într-o clasa separată fata de restul programului pentru a asigura modularitate si pentru a ușura mentenanța sistemului.Design-ul clasei se va face dupa principiile modelulde design "Single Tone ".

Clasa va contine o metoda pentru a generarea sunetului, care va permite ca sunetul generat sa poata fi modificat în funcție e necesitățiile aplicației. De asemenea clasa va contine un contructor si un destructor.

Metoda care va genera undele sonore este "**tone()" .**

Descrierea funcției tone() :

Generează un semnal dreptunghiular pe o frecvență specifica (cu factorul de umplere de 50%) pe un pin . Durată poate fi specificată, în caz contrar semnalul va continua până la un apel la noTone (). Pinul poate fi conectat la un buzzer piezo sau la alt difuzor pentru a reda tonuri.  
 Nu este posibilă generarea semnalelor cu o frecventa mai mica de 31Hz.   
  
*Sintaxă*  
 tone (PIN, frecvență)  
 tone(PIN, frecvență, durată)  
   
Parametrii  
 pin: pinul pe care se generează tonul  
 frecvența: frecvența tonului în hertz   
  
Durata: durata tonului în milisecunde (opțional) - nesemnată lungă  
 Funcția nu returnează nici o valoare .

Pentru a generarea tonurilor sonore se va folosi funcția **“noTone()”** :

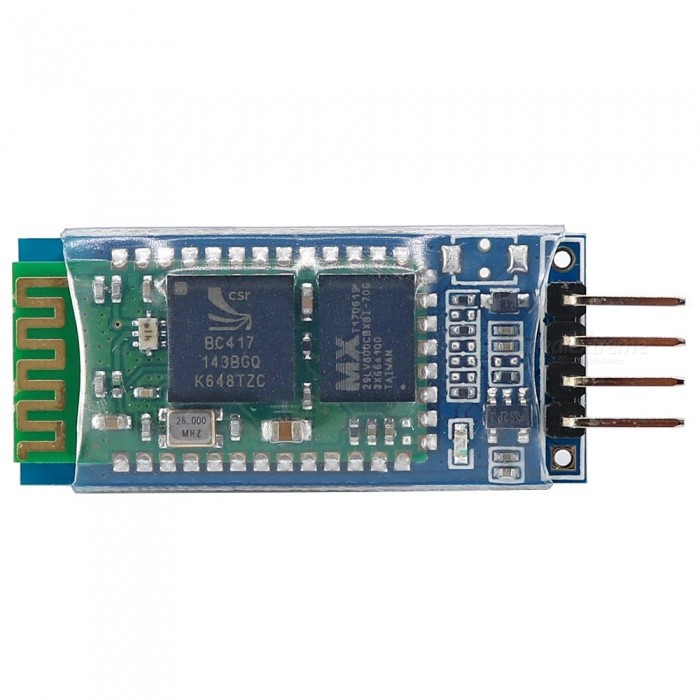
Descriere  
 Oprește generarea semnalului dreptunghiular declanșat de tone(). Nu are efect dacă nu se generează tonuri.  
   
Sintaxă  
 noTone (pin)  
   
Parametrii  
 pin: pinul pe care să se oprească generarea tonului  
   
Funcția nu returnează nici o valoare.

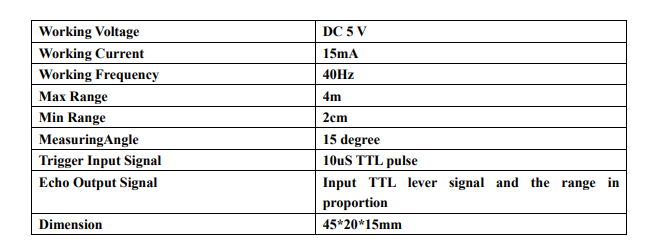
**Moldulul Bluetooth**

***-Hardware:***

Requirements

* Mudul Bluettoth Hc – 06



* Este conectat direct la placuta Arduino si controlata de catre procesorul aceteia
* Scop: facilitare comunicării dintre plăcuța Arduino și una Raspberry PI
* Are rol bidirectional: transmitere de date (sunt transmise spre un modul Raspberry PI informații de distanțele măsurate prin intermediul senzorilor ultrasonici) și primire de comenzi
* Este integrată în sistemul de parcare împreună cu plăcuța Arduino, senzorii pentru parcare și buzzer-ul
* Cele ai importante caracteristici electrice ale acestui modul sunt prezentate în tabelul de mai jos: 
* Frecvența la care se vor transmite datele măsurate de senzori către plăcuța Raspberry PI va fi stabilită de comun acord cu reprezentații produsului/stake holders.

Documentație

Datasheet-ul acestui modul este disponibil in următorul link:

<http://www.instructables.com/id/Tutorial-Using-HC06-Bluetooth-to-Serial-Wireless-U/>

Conexiunea la placuța Arduino a modului se va realiza conform indicațiilor și restricțiilor regăsite în Datasheetul acestuia.

Transmisia și primirea datelor se va asigura prin intermediul programelor în limbajul C/C++, în cadrul Arduino IDE. Detaliile software de comunicare se regăsesc la secțiunea software de mai jos.

Modului Bluetooth dispune de 4 pini: 2 pentru alimentare, unul pentru transmitrea datelor (TX) și unul pentru primirea datelor (RX). Transmiterea și trimiterea datelor se face în mod serial.

Frecvența de transmisie și de primirea a datelor poate fi modificată software în cadrul aplicației.

***-Software***

Requirements

* Programare și configurarea acestui modul se va face folosind o platforma C/C++, în cadrul Arduino IDE
* Funcționarea, configurarea, trimiterea și primirea de date vor fi asigurate de funcții special definite într-un fișier special dedicat acestui modul
* Scopul principal al modului fiind transmiterea și primirea de Frame-uri
* Frame-ul (structură definită pentru facilitarea unei transmisii corecte către dispozitivul Raspberry) în care se vor transmite datele va fi stabilit ulterior
* Structura de cod dedicată modului bluetooth va conține doar funcții specifice pentru schimbul de date, urmând ca prelucrarea acestora să parte a sarcinilor unui controller la nivelul sistemului de calcul al Arduino

Documentație

Din punct de vedere al designului de programare, modulul Bluetooth va fi realizat prin intermediul unei clase de tip Singleton (se asigură prin aceasta existența unui singur obiect de tip Bluetooth la nivelul aplicației). Aceasta presupune ca toți constructorii să aibă un modificator de acces „private”. Posibilitatea instanțierii unui obiect se va realiza prin intermediul metodei getInstance().

Atributele care se vor utiliza în cadrul clasei sunt: o variabila privată de tip int, numită „dateTransmise”, în care se vor reține ultimele date transmise prin intermediul modului Bluetooth, și o variabilă de tip vector de int „pini”, utilizată pentru configurarea modului pe plăcuța Arduino.

Metodele „construireFrame()” și „decodareFrame()” vor fi utilizate pentru construirea Frameurilor care urmează să fie transmise la Raspberry PI, conform unui model prestabilit, respectiv pentru decodificarea comenzilor primite de la plăcuța Raspberry, în legătură cu volumul maxim care poate fi redat de către componenta de avertizare.

„trimiteDate()” și „primesteDate()” sunt metodele care vor asigura transmiterea și primirea de informații către și de la modul Raspberry PI. Acestea se vor ocup atât cu transmiterea eectivă a informațiilor, cât și cu codificarea, respectiv decodificarea informațiilor. Metoda „transmite Date()” va returna o valoare în funcție de succesul sau insuccesul operație de transmitere.

Există, de asemenea, posibilitatea distrugerii obiectului creat prin intermediul funcției distrugeBluetooth(). Funcțiile init() și dinit() sunt utilizate la construirea, respectiv didtrugerea unui obiec de tipul Bluetooth.

**Senzor ultrasonic HC-SR04**

**Requirements**

In acest proiect, senzorul ultrasonic HC-SR04 va avea rolul de a masura distanta de la masina, pe bara careia vor fi amplasat, pana la un posibil obiect. Vom avea nevoie de 5 senzori pentru a oferi soferului o perspectiva clara a asupra locației mașinii în raport cu obiectul.

Avand un unghi de masurare de 15 grade, senzorii vor forma conuri de detectie. Senzorii trebuie asezati pe bara din spate a masinii, astfel incat conurile de detectie sa se intersecteze la distanta de 150 cm.

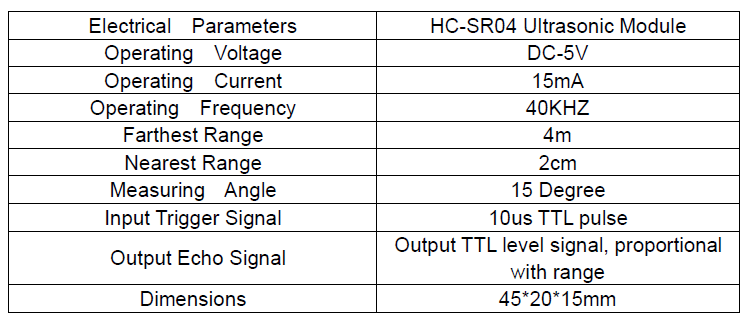
**Hardware**

Senzorul ultrasonic HC-SR04 este prezentat in figura 1.



Fig. 1 Senzorul ultrasonic HC-SR04

Principalele caracteristici ale senzorului sunt prezentate in tabelul de mai jos:



Dupa cum putem observa in figura 1, senzorul ultrasonic are 4 pini: Ground, VCC, Trig si Echo. Pinii Trig si Echo trebuie conectati la orice pin Digital I/O al placii Arduino.

**Software**

Pentru configurarea acestui modul, vom folosi limbajul C/C++, compilatorul fiind Arduino IDE.

Vom crea clasa Senzor care va avea 5 parametrii de tip int: distantaActuala, idSenzor, MIN\_DISTANTA, MAX\_DISTANTA si pini. Vom seta intervalul de masurare intre 15 cm si 150 cm.

Metoda citesteDistanta() ne va furniza distanta de la senzor pana la un posibil obiect. Pentru a putea masura aceasta distanta, va trebui sa setam pinul Trig ca find High, iar apoi sa observam cat timp pinul Echo este activ. Acesta este activ de la momentul emiterii undei de catre pinul Trig pana la intoarcerea acesteia inapoi la senzor. Pentru a afla perioada aceasta vom folosi functia pulseIn(). Iar distanta in centimetri se va calcula impartind perioada cat timp pinul Echo este High la 29, apoi la 2. Viteza sunetului este 343 m/s => 29.155 µs/cm ~= 29. Iar 29 il impartim din nou la 2 pentru ca sunetul trebuie sa traverseze distanta de doua ori.

Aceasta clasa va avea atat un constructor cat si un destructor.

***Raspberry Pi 3***

Raspberry Pi 3 Model B este primul model din a treia generatie de Raspberry Pi. A inlocuit Raspberry Pi 2 Model B in Februarie 2016

In proiect, Rasperry Pi 3 va avea rolul de a primi/transmite date de la Arduino prin bluetooth. Datele primite vor fi prelucrate pentru interfata grafica, iar cele trimise la Arduino vor fi pentru buzzer.



**Requirements:**

1. Trebuie sa se conecteze la arduino prin bluetooth
2. Comunicare date arduino – raspberry si invers
3. Prelucrare date pentru interfata grafica
4. Scripturi pentru conectare automata

**Hardware:**

* Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
* 1GB RAM
* BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
* 100 Base Ethernet
* 40-pin extended GPIO
* 4 USB 2 ports
* 4 Pole stereo output and composite video port
* Full size HDMI
* CSI camera port for connecting a Raspberry Pi camera
* DSI display port for connecting a Raspberry Pi touchscreen display
* Micro SD port for loading your operating system and storing data
* Upgraded switched Micro USB power source up to 2.5A

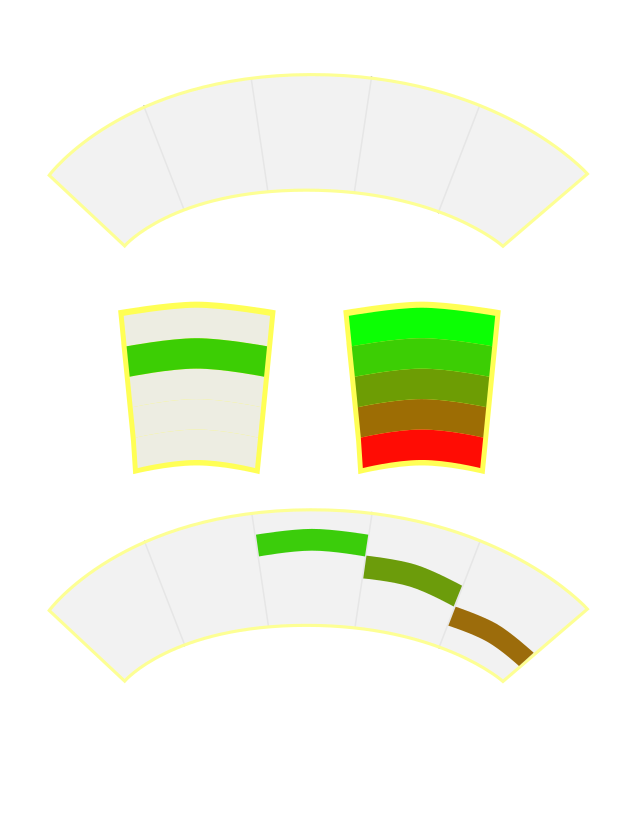
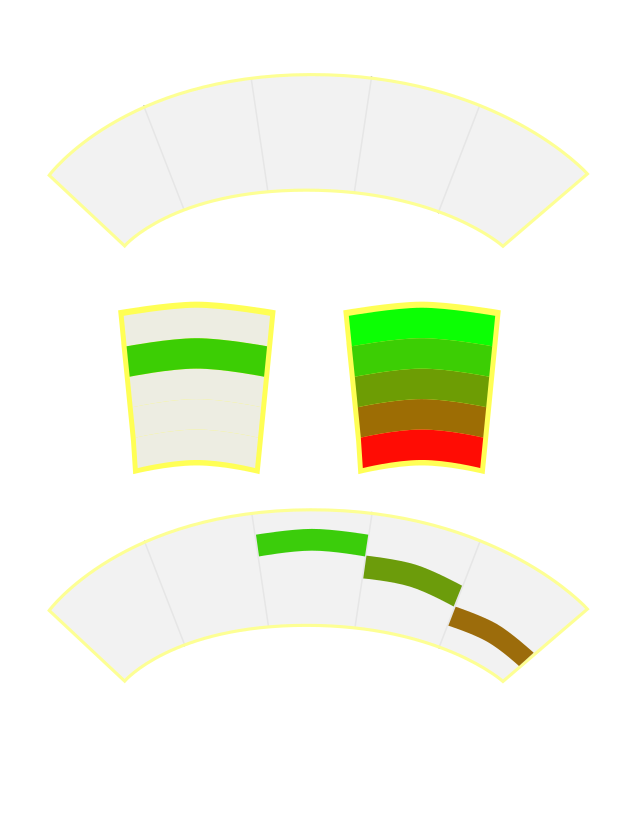
**Software**

Vom implementa un script în python care va automatiza procesul de primire a datelor de la Arduino. Vom folosi și Linux în paralel pentru a scrie cod și a realiza conexiunea bluetooth Arduino – Raspberry. Datele vor trece printr-un algoritm prin care să fie pregătite pentru a merge către Interfața grafică, care va fi realizată cu QT, dar și înapoi către Arduino pentru oferi informații buzzer-ului.

**Interfata grafica**

**Requirements**

1. Trebuie sa fie reprezentati toti cei 5 senzori.
2. Un senzor este impartit in 5 zone.
3. Nu exista distanta intre zone sau intre senzori.
4. Doar o singura zona a unui senzor poate fi vizibila la un moment dat.
5. Cand in fata unui senzor exista un obiect ( la mai putin de 150 cm ) zona corespunzatoare distantei obiectului fata de masina trebuie sa se coloreze ( restul zonelor raman invizibile).
6. Zonele sunt colorate de la verde ( cel mai aproape de masina ) la rosu ( cel mai departe de masina ).
7. Update-ul senzorilor se face o data la 30 de secunde.
8. Cand frana masinii este trasa ( masina nu este in miscare ) zonele vor fi colorate in alb.\*



**Documentatie**

Pentru interfata grafica se folosesc 2 clase.

**Clasa Senzor**

- folosita pentru a crea obiecte de timpul Senzor.

Atribute private:

DIST\_MAX – distanta maxima la care se poate afla un obiect fata de masina pentru a putea fi detectat de senzor. Este setat initial cu 150 cm.

NR\_SEGM – numarul de zone in care este impartit un senzor. Setat initial cu 5. DIST\_ACTUALA – distanta actuala a unui obiect fata de senzor. std::vector<QPainter> segments – vector in care sunt stocate arcele care reprezinta zonele segmentului.

Functii publice: Senzor(const int DIST\_MAX, const int NR\_SEGM) – constructor care primeste distanta maxima si numarul de segmente. In cazul in care clientul se razgandeste in privinta distantei maxime a unui senzor sau a numarul de zone acestea pot fi alese la constructia senzorilor. int **getNR\_SEGN**() constint **getDIST\_MAX**() const - getter-re pentru atributele care raman fixe ale senzorului. O data creat unui senzor nu I se poate modifica distanta maxima sau numarul de zone.int **getDIST\_ACTUALA**() const void **setDIST\_ACTUALA**(const int DIST\_ACTUALA)- getter + setter pentru distanta actuala. Aceasta se va schimba la intervalul de timp specificat de client. void **drawSenzor**() - deseneaza senzorul. Asta presupune ca vectorul segmente a senzorului va fi populat cu NR\_SEGM de arce, initial setate pentru a fi transparente. void **updateZona**() – coloreaza zona corespunzatoare distatei actuale. Apeleaza functia **getDIST\_ACTUALA**() daca aceasta apartine intervalului [0,150], map-eaza acest numar la o zona a senzorului. O daca ce stim ce segment trebuie colorat il coloram corespunzator ( pe range-ul de culori stabilit ).**Clasa MainWindow** FREQ - intervalul de timp la care se modifica distanta actuala a senzorului. std::vector<Senzor> senzori; - vectori cu cei 5 senzori.void **getFREQ**() const; void **setFREQ**(const int FREQ);- getter + setter pentru frecventa. Poatea fi modificata la preferinta clientului. void **updateSenzor**(int senzor\_id, int senzor\_dist) – senzorului cu id-ul cu senzor\_id i se face update la distanta actuala. Aceasta procedura este apelata la intervalul de timp stabilit de FREQ. In interiorul ei este apelata si functia **updateZona**() a senzorului. void **receiveDistante**() - primeste distantele pentru fiecare senzor si seteaza inautru functiei aceste distante pentru fiecare senzor. void ***paintEvent***(QPaintEvent \*e) – folosita pentru desenarea senzorilor.

In main se vor construi cei 5 senzori iar mai apoi intr-un loop care face o iteratie o data la 30 de sec se vor:

- extrage datele necesare, adica cele 5 numere care reprezinta distantele unor obiecte fata de senzori.

- face update la toti senzorii.

Loop-ul este oprit de un semnal.

**Design**

