**ROMÂNIA**

**MINISTERUL APĂRĂRII NAȚIONALE**

**ACADEMIA TEHNICĂ MILITARĂ ,,FERDINAND I”**

**FACULTATEA DE SISTEME INFORMATICE ȘI SECURITATE CIBERNETICĂ**

**Specializare: Calculatoare și Sisteme Informatice pentru Apărare și Securitate Națională**



**Platformă autonomă care să se deplaseze pe un traseu delimitat și să ocolească obstacolele**

Realizator,

**Std. Plt Maj. Hariga George-Codrin**

Îndrumator,

**Conf. dr. ing Laurențiu Mărgărit**

**București**

**2021**

**Cuprins**

[**Abstract** 9](#_Toc74266586)

[**Rezumat** 10](#_Toc74266587)

[**I.** **Introducere** 11](#_Toc74266588)

[**1.** **Descrierea Proiectului** 11](#_Toc74266589)

[**2.** **Alegerea Proiectului** 11](#_Toc74266590)

[**3.** **Istoric** 12](#_Toc74266591)

[**4.** **Mediu de lucru** 12](#_Toc74266592)

[**II.** **Configurație generală** 14](#_Toc74266593)

[**1.** **Platforma FRDM-KL25Z** 14](#_Toc74266594)

[**2.** **Servomotor** 15](#_Toc74266595)

[**3.** **Motoare** 15](#_Toc74266596)

[**4.** **Modulul LineScanCamera** 15](#_Toc74266597)

[**5.** **Senzori ultrasunet** 16](#_Toc74266598)

[**7.** **XBee** 17](#_Toc74266599)

[**8.** **Senzor magnetic cu efect Hall** 19](#_Toc74266600)

[**9.** **Platforme ajutătoare** 19](#_Toc74266601)

[**III.** **Configurația hardware** 20](#_Toc74266602)

[**1.** **Platforma amplificatoare** 20](#_Toc74266603)

[**2.** **Platforma senzorilor de distanță** 24](#_Toc74266604)

[**3.** **Conectarea tuturor platformelor** 26](#_Toc74266605)

[**IV.** **Implementarea software-ului** 27](#_Toc74266606)

[**1.** **Configurație generală** 27](#_Toc74266607)

[**2.** **Driver GPIO** 27](#_Toc74266608)

[**3.** **Driver UART** 28](#_Toc74266609)

[**4.** **Driver TPM** 31](#_Toc74266610)

[**5.** **Driver PIT** 37](#_Toc74266611)

[**6.** **Driver ADC** 39](#_Toc74266612)

[**7.** **Menținerea drumului** 42](#_Toc74266613)

[**8.** **PID** 44](#_Toc74266614)

[**9.** **Ocolirea obstacolelor** 46](#_Toc74266615)

[**V.** **Testare** 48](#_Toc74266616)

[**1.** **Interacțiunea cu platforma** 48](#_Toc74266617)

[**2.** **Diagrame UML** 48](#_Toc74266618)

[**3.** **Testarea modulelor** 51](#_Toc74266619)

[**VI.** **Concluzii** 55](#_Toc74266620)

[**1.** **Probleme întampinate** 55](#_Toc74266621)

[**VII.** **Bibliografie** 58](#_Toc74266622)

[**VIII.** **Anexe** 59](#_Toc74266623)

[**1.** **Anexa 1** 59](#_Toc74266624)

[**2.** **Anexa 2** 63](#_Toc74266625)

[**3.** **Anexa 3** 65](#_Toc74266626)

[**4.** **Anexa 4** 69](#_Toc74266627)

[**5.** **Anexa 5** 70](#_Toc74266628)

[**6.** **Anexa 6** 74](#_Toc74266629)

[**7.** **Anexa 7** 74](#_Toc74266630)

[**8.** **Anexa 8** 76](#_Toc74266631)

[**9.** **Anexa 9** 76](#_Toc74266632)

Abrevieri

|  |  |  |
| --- | --- | --- |
| Nr. | Abreviere | Semnificație |
| 1 | AD | Analog to digital |
| 2 | ADC | Analog to digital converter |
| 3 | AO | Analog output |
| 4 | CLK | Clock |
| 5 | DIY | Do it yourself |
| 6 | FRDM | Freedom |
| 7 | GND | Ground |
| 8 | GPIO | Genral purpose input output |
| 9 | IDE | Integrated development environment |
| 10 | LSB | Least semnificant bit |
| 11 | MSB | Most semnificant bit |
| 12 | MUX | Multiplexer |
| 13 | OSR | Over sampling ratio |
| 14 | PCB | Printed circuit board |
| 15 | PID | Proporțional, integrativ, derivativ |
| 16 | PIT | Periodic interrupt timer |
| 17 | PTXn | Port X pin n |
| 18 | PWM | Pulse-width modulation |
| 19 | RX | Reciever |
| 20 | SAR | Successive approximation register |
| 21 | SI | Serial input |
| 22 | SIM | System integration module |
| 23 | SRAM | Static random-access memory |
| 24 | TPM | Timer/PWM module |
| 25 | TRIG | Trigger |
| 26 | TX | Transmitter |
| 27 | UART | Universal asynchronous receiver-transmitter |
| 28 | UML | Unified modeling language |
| 29 | VCC | Voltage common colector |
| 30 | VDD | Voltage drain drain |
| 31 | VIN | Voltage input |
| 32 | VOUT | Voltage output |
| 33 | VREF | Voltage reference |

Figuri

[Fig. 1 FRDM-KL25Z 14](#_Toc74266694)

[Fig. 2 Caracteristici semnale cameră 16](#_Toc74266695)

[Fig. 3 Senzorul HC-SR04+ 17](#_Toc74266696)

[Fig. 4 Modulul XBee 17](#_Toc74266697)

[Fig. 5 XBee Explorer Regulated 18](#_Toc74266698)

[Fig. 6 XBee Explorer USB 18](#_Toc74266699)

[Fig. 7 Senzor magnetic cu efect Hall 19](#_Toc74266700)

[Fig. 8 M74HCT244B1 20](#_Toc74266701)

[Fig. 9 74157N 21](#_Toc74266702)

[Fig. 10 Schemă platforma amplificatoare 22](#_Toc74266703)

[Fig. 11 Poză platforma amplificatoare 22](#_Toc74266704)

[Fig. 12 Schemă platforma senzorilor de distanță 24](#_Toc74266705)

[Fig. 13 Poză platforma senzorilor de distanță 25](#_Toc74266706)

[Fig. 14 Semnal UART 29](#_Toc74266707)

[Fig. 15 Schema transmițătorului UART 30](#_Toc74266708)

[Fig. 16 Schema receptorului UART 31](#_Toc74266709)

[Fig. 17 Schema modulului TPM 32](#_Toc74266710)

[Fig. 18 TPM Up counting 33](#_Toc74266711)

[Fig. 19 TPM Up-Down counting 33](#_Toc74266712)

[Fig. 20 TPM generare PWM 35](#_Toc74266713)

[Fig. 21 Semnale LineScanCamera 38](#_Toc74266714)

[Fig. 22 Schema modulului ADC 39](#_Toc74266715)

[Fig. 23 SAR pe 4 biți 40](#_Toc74266716)

[Fig. 24 Măsurătoare cameră 41](#_Toc74266717)

[Fig. 25 Măsurătoare cameră cu indicații ale zonelor 42](#_Toc74266718)

[Fig. 26 Schemă PID 44](#_Toc74266719)

[Fig. 27 PID, caracteristicile unui semnal 45](#_Toc74266720)

[Fig. 28 Semnalele TRIG și ECHO 47](#_Toc74266721)

[Fig. 29 Diagrama Use Case 48](#_Toc74266722)

[Fig. 30 Diagrama de activități 50](#_Toc74266723)

[Fig. 31 Turația Matlab 52](#_Toc74266724)

[Fig. 32 Turația Matlab medie 53](#_Toc74266725)

[Fig. 33 Senzor distanță centru Matlab 54](#_Toc74266726)

[Fig. 34 Siguranța termică F2 56](#_Toc74266727)

[Fig. 35 Siguranța termică F1 57](#_Toc74266728)

Tabele

[Tabel 1 Configurație platformă amplicatoare 23](#_Toc74266729)

[Tabel 2 Configurație platforma senzorilor de distanță 25](#_Toc74266730)

[Tabel 3 TPMx\_CnSC Edge, level și mod de funcționare canal 34](#_Toc74266731)

# **Abstract**

This project aims to create an autonomous platform that travels on a delimited route with the possibility of avoiding obstacles that may arise along the way. The components used in the NXP Cup competition (formerly known as the Freescale Cup) are largely used.

The first chapter has an introductory role, where the project is described and its purpose, the choice of the project is described, a history of similar projects, which currently exists on the market and the working environment in which the source code for the project is made.

The second chapter has the role of presenting each component in general, their operating parameters, what role and what functionalities they have.

The third chapter presents the hardware configuration, how the development platform and auxiliary platform modules are physically linked. The amplifier platform and that of the distance sensors, their electrical diagrams and their functionality are also presented.

The fourth chapter has the role of presenting the software implementation. The implemented drivers and algorithms developed for line detection and avoiding obstacles are presented.

The fifth chapter presents the way in which the debugging was done, what tests were done to calibrate the system and to verify the functionalities. This chapter also contains UML diagrams to facilitate understanding of the algorithms and how to interact with the platform.

The sixth chapter presents the conclusions drawn and the problems encountered along with possible solutions.

# **Rezumat**

Acest proiect de licență își propune realizarea unei platforme autonome care să se deplaseze pe un traseu delimitat cu posibilitatea de a ocoli obstacolele care pot apărea pe parcurs. Se folosesc, în mare, componentele folosite la competiția NXP Cup (cunoscută in trecut ca și Freescale Cup).

Primul capitol are rol introductiv, unde, este descris proiectul și scopul acestuia, este descrisă alegerea proiectului, un istoric despre proiecte asemănătoare, ce există la ora actuală pe piață și mediul de lucru în care este realizat codul sursă pentru proiect.

Al doilea capitol are rolul de a prezenta fiecare componentă la modul general, parametrii lor de funcționare, ce rol au și ce funcționalități au.

Al treilea capitol prezintă configurația hardware, cum sunt legate fizic modulele de platforma de dezvoltare și de platformele auxiliare. Sunt de asemenea prezentate platforma amplificatoare și cea a senzorilor de distanță, schemele lor electrice și ce funcționalitate au.

Al patrulea capitol are rolul de a prezenta implementarea software. Sunt prezentate driverele implementate și algoritmii dezvoltați pentru detecția liniei, menținerea traseului și ocolirea obstacolelor.

Capitolul al cincilea prezintă modul în care s-a făcut debug, ce teste s-au realizat pentru a calibra sistemul și pentru a verifica funcționalitățile. Acest capitol conține și diagrame UML pentru a facilita înțelegerea algoritmilor și a modului de interacțiune cu platforma.

Capitolul al șaselea prezintă concluziile trase și problemele întâmpinate împreună cu posibilele soluții.

# **Introducere**

## **Descrierea Proiectului**

Platforma autonomă se deplasează pe traseul marcat cu ajutorul a două motoare de curent continuu de 7.2V, iar direcția este controlată de un servomotor MG995. Pentru a putea detecta linia traseului este folosit un modul LineScanCamera de la Texas Advanced Optical Solutions (128x1 pixeli), iar pentru a evita obstacolele se folosesc trei senzori pe bază de unde ultrasonice, model HC-SR04+, pentru a luat date dintr-un unghi mai larg. Plăcuța de dezvoltare este un FRDM-KL25Z cu un microprocesor ARM Cortex-M0+ cu 128KB de memorie flash și 16KB de SRAM, produsă de NXP. Toate acestea vor fi conectate la o baterie de 8V și 2500mAh și puse pe mașinuța din kitul de la NXP. Proiectul va putea fi folosit ca o baza a unor software-uri, mai complexe, ce vor fi folosite în scenarii reale. Să luam ca exemplu un autoturism și pilotul automat al acestuia. Autovehiculul ar putea folosi funcționalitatea proiectului de parcurgere a traseului, menținând banda, având ca delimitator marcajele de pe drum. Un alt lucru important pe care l-ar putea face pilotul automat, este sa evite obstacolele, cum ar fi: pietoni, gropi, alte mașini sau orice obiecte care ar putea avaria mașina sau care ar putea pune în pericol viața cuiva. Avantajul ocolirii automate este că are timp de reacție mai bun și poate vira mașina mai optim și mai bine decât un om ar putea, dacă este implementat totul corect, și astfel s-ar putea evita un număr considerabil de accidente. O altă aplicabilitate ar putea fi trimiterea unei astfel de platforme într-o zonă de risc pentru oameni cum ar fi o zonă cu radiații unde un om ar putea fi afectat.

## **Alegerea Proiectului**

Proiectul își propune să eficientizeze timpii de răspuns ai platformei autonome și mărirea preciziei în evitarea obstacolelor prin intermediul unor date de natură optică și acustică. Un alt obiectiv important al acestui proiect este reprezentat de reducerea cât mai mult a consumului de energie electrică fără a afecta capabilitățile sistemului automat.

## **Istoric**

Creearea unei platforme autonome cu aceste funcționalități nu este ceva nou. Spre exemplu, NXP Cup organizeaza anual un concurs pe această temă, cu proba de traseu (unde contează timpul de parcurgere și energia consumată), proba de obstacole (ocolirea obstacolelor), proba traseului în forma de opt și o alta probă unde mașina trebuie sa fie capabilă să detecteze semnele de circulație. Pentru a construi o astfel de platformă, participanții foloseau în general o placuță de dezvoltare FRDM-KL25Z, un shield FRDM-TFC, două motoare de curent continuu, un servomotor, un LineScanCamera, o baterie și în funcție de proba la care participau alți senzori. Ca mediu de dezvoltare pentru acest concurs se folosesc MCUXpress, Keil µVision, CodeWarrior, Mbed Studio, Matlab/Simulink. Alte proiecte asemănatoare au fost realizate și cu alte placuțe de dezvoltare precum Arduino Uno sau Raspberry Pi, însă microcontrollerul de la Arduino (Atmega328p) nu este la fel de rapid, iar Raspberry Pi este cu mult mai puternică decât este necesar și are prin urmare un consum mai mare de energie. Putem da un exemplu, din viața cotidiană a aplicării funcționaltităților acestui proiect și anume, tehnologia Lane assist. Aceasta folosește datele de la o cameră atașată la oglinda retrovizoare pentru a menține autovehiculul pe banda sa. Un alt exemplu este funcția Summon de la autoturismele Tesla prin care, mașina poate veni singura către proprietar ocolind obstacolele (precum alte mașini).

## **Mediu de lucru**

Mediul de lucru folosit pentru acest proiect este Keil µVision 5. Structura unui proiect în acest IDE este următoarea:

* Directorul parinte care denumirea de forma Project: <nume proiect>;
  + Target 1, numele implicit al directorului creat de IDE, acesta va fi modificat în Licenta pentru proiectul acesta;
    - Source Group 1, acesta initial este gol și aici se vor crea toate sursele necesare, denumirea va fi modificată în src pentru acest proiect;
    - inc, este un grup creat ulterior care conține toate fișierele de tip header folosite pentru dezvoltarea proiectului.
  + Următoarele fișiere vor conține surse generate de mediul de dezvoltare în funcție de ce module sunt introduse de către utilizator.

# **Configurație generală**

## **Platforma FRDM-KL25Z**

Această platformă conține un microprocesor ARM Cortex-M0+ cu frecvența de 48Mhz, are 128KB memorie flash, 16KB memorie SRAM si 80 de pini. Frecvența magistralei și frecvența flash-ului sunt jumătate din cea a procesorului și anume 24Mhz (De reținut că aceste frecvențe sunt maxime și placuța poate fi folosită la frecvențe mai mici). De asemenea mai conține doi senzori, un accelerometru MMA8451Q, un touch ,,slider” capacitativ și un LED RGB. Pentru a se putea depana pe această placuță se va folosi interfața OpenSDA (Portul se poate vedea în figura 1). Avantajul acestei platfome este că funționează la tensiuni de alimentare mici și are consum de energie electrică redus. Placuța este Mbed enabled și suportă Zephyr OS.



Fig. 1 FRDM-KL25Z

## **Servomotor**

Servomotorul este un MG995. Acesta are rolul de a schimba direcția de mers a platformei mobile. Servomotorul va fi conectat la GND, la 5V și va primi, de la placuța de dezvoltare, un semnal PWM (Pulse Width Modulation). Semnalul va avea frecvența de 50Hz (sau perioada 20ms), iar factorul de umplere va fi de la 2.5% (poziția minimă) la 12.5% (poziția maximă), toate celelalte unghiuri aflându-se între aceste două valori. Pentru a genera semnalul PWM, se va folosi modulul TPM (Timer PWM Module) al placuței KL25Z.

## **Motoare**

Motoarele au rolul de a transforma energia electrică în energie mecanică si astfel, să deplaseze platforma. Pentru a putea fi folosite, vor fi cuplate la un driver de motoare, o punte H, care va primi ca input semnalele PWM (care sunt generate tot cu ajutorul unui TPM ca la servomotoare) generate de plăcuță și vor transmite un curent electric, la motoare, cu valori între -7.2V si 7.2V (Valorile negative sunt pentru a schimba sensul de rotație). Motivul pentru care este necesar un driver de motoare este faptul că, dacă conectăm motoarele direct la placuța de dezvoltare, există o posibilitate mare să se ardă circuitele acesteia sau să nu aibă puterea necesară pentru a se deplasa.

## **Modulul LineScanCamera**

Acest modul poate produce o image de 128x1 pixeli și poate fi vazută ca un semnal pe care platforma îl va folosi pentru a detecta forma traseului. Pentru aceasta, camera are la puși dispoziție cinci pini. Pinul SI (Serial Input) are rolul de a începe o scanare. Pinul CLK va primi ca input un semnal de ceas generat de placuța de dezvoltare. Prin intermediul pinului AO (analog output) se vor transmite pixelii ca valori analogice ce vor fi furnizați modulului ADC al platformei. Pinii VDD și GND sunt folosiți pentru a alimenta modulul. Dacă se crește frecvența ceasului prea mult, riscăm ca pixelii să nu aibă timp destul de expunere, iar dacă setăm frecvența prea mica, pixelii vor fi supraexpuși. De aceea trebuie sa găsim un echilibru pentru aceste valori. Modul de operare a acestui modul este reprezentat în diagrama următoare:



Fig. 2 Caracteristici semnale cameră

## **Senzori ultrasunet**

Pentru a detecta obstacolele vom folosi trei senzori de distanță HC-SR04+. Acest tip de senzor are patru pini și anume, VCC și GND pentru alimentare, TRIG, pin de input prin care putem comunica modulului să înceapă o măsurătoare și ECHO, pin de output prin intermediul căruia vom calcula distanța propriu-zisă. Modulul are următorul principiu de funcționare:

* Se transmite un impuls cu valoarea HIGH timp de 10us pe pinul TRIG.
* Modulul trimite opt unde dreptunghiulare la o frecvență de 40KHz.
* Pinul Echo este setat pe HIGH.
* Se așteaptă întoarcerea undelor sonore, moment în care pinul Echo este setat pe LOW.

Senzorul poate măsura distanțe de la 2cm până la 450cm în funcție de tensiunea de alimentare.

Tensiunea de alimentare poate fi și de 5V și de 3.3V, dar pentru acest proiect, vom alimenta modulul la 3.3V întrucât nu avem nevoie de distanțe mai mari de 400cm. Pinii sunt prezentați în figura 3.



Fig. 3 Senzorul HC-SR04+

## **XBee**

Pentru a putea realiza o depanare de la distanță și pentru a primi date de la platformă (informații precum loguri) vom folosi două module XBee care ne asigură o comunicație wireless. Aceste module au la bază protocolului 802.15.4, comunică la o frecvență de 2.4Ghz și sunt alimentate la o tensiune de 5V. Pentru a transmite date și pentru a primi date de la aceste module se va folosi interfața UART cu un baud rate de 9600. Figura 4 prezintă pinii acestor module.



Fig. 4 Modulul XBee

Modulul atașat platformei este folosit cu ajutorul unui XBee Explorer Regulated, produs de Sparkfun (Fig. 5). Astfel, modulul va avea doar patru pini care vor folosiți și anume: VCC, GND, TX, RX. Primii doi sunt folosiți pentru alimentarea modulului, iar RX și TX sunt folosiți pentru comunicarea propriu-zisă prin UART. Pentru modulul care va fi conectat la distanță, vom folosi un XBee Explorer USB produs tot de Sparkfun pentru a-l putea conecta la un laptop.



Fig. 5 XBee Explorer Regulated

**

Fig. 6 XBee Explorer USB

## **Senzor magnetic cu efect Hall**

Pentru a putea măsura viteza platformei s-au lipit pe interiorul roții din partea din spate dreapta șase magneți care ne vor ajuta să măsurăm turația. Vom folosi un senzor magnetic cu efect Hall model Honeywell alimentat la 3.3V. Acesta conține trei pini, VCC, GND și ieșirea. Ieșirea este de tip active low, iar acest lucru înseamnă că senzorul ne va furniza o tensiune de 0V în momentul în care magnetul este aliniat cu senzorul și invers în momentul în care magnetul nu mai este aliniat. În figura 7 se pot vedea pinii senzorului.



Fig. 7 Senzor magnetic cu efect Hall

## **Platforme ajutătoare**

Pentru a putea comanda motoarele și servomotoarele avem nevoie de o tensiune de 5V, însă semnalele provenite de la FRDM-KL25Z sunt de 3.3V. Una din platformele ajutătoare amplifică semnalele provenite de la placuța de dezvoltare la 5V și acționează ca o interfață între placută și driverul de motoare.

O altă problemă este poziționarea senzorilor de distanță, iar ca o soluție la această problemă s-a construit o mică platformă care să acționeze ca o interfață între placuța de dezvoltare și senzori și pe care aceștia să fie poziționați.

# **Configurația hardware**

În acest capitol vom vorbi despre cum sunt legate fizic modulele între ele, și vor fi explicate circuitele logice auxiliare folosite. Pentru realizarea schemelor electrice ale PCB-urilor s-a folosit o aplicație denumită DIY Layout Creator.

## **Platforma amplificatoare**

Această placuță PCB are rolul de a amplifica semnalele provenite de la placuța de dezvoltare. Pentru amplificarea semnalului, s-a folosit un amplificator model M74HCT244B1 (Fig. 8) alimentat la 5V.



Fig. 8 M74HCT244B1

Pinii acestui circuit logic sunt codificați astfel:

* Pinii care sunt de forma nAn sunt intrările.
* Pinii care sunt de forma nYn sunt ieșirile.
* Pinii care sunt de forma nG sunt pinii de enable.
* Pinii VCC și GND sunt pentru alimentare.

Întrucât platforma de dezvoltare trebuie să poată schimba sensul de rotație al motoarelor, iar platforma pentru controlul motoarelor nu are funcționalitatea de a inversa polaritatea semnalul primit, a fost necesară introducerea unui multiplexor model 74157N (Fig. 9), alimentat la 5V.



Fig. 9 74157N

Pinii acestui circuit logic sunt codificați astfel:

* Perechile de pini An, Bn sunt semnalele ce vor fi multiplexate.
* Pinii care sunt de forma Yn sunt ieșirile pentru semnalele multiplexate.
* Pinul G este pinul de enable.
* Pinul S este pinul de selecție al semnalului.
* Pinii VCC și GND sunt pentru alimentare.

Astfel, având acest multiplexor, putem inversa polaritatea semnalului PWM furnizat de amplificator și implicit, putem schimba sensul de rotație al motoarelor.

Pentru a asigura o legătură mai ușoară între senzorul de turație, cameră, XBee și plăcuța de dezvoltare, au fost lipiți pini adiționali pe această placută PCB conform figurii numărul 10 care este prezentată mai jos. Figura numărul 11 este plăcuța amplificatoare fotografiată în realitate.



Fig. 10 Schemă platforma amplificatoare

**

Fig. 11 Poză platforma amplificatoare

Pinii pentru cameră, pentru XBee și pentru senzorul de turație nu trec prin circuite logice intermediare. Pinii de la coordonatele JC, JD, KC, KD, PC, PD, QC, QD (Fig. 10) nu sunt legați la nimic și au doar rolul de suport mecanic pentru modulul XBee. Pinii sunt dublați în unele locuri pentru a ușura realizarea conexiunilor fizice.

În următorul tabel avem prezentate toate conexiunile dintre placuța de dezvoltare, placuța amplificatoare și cupla pentru platforma care controlează motoarele, unde este cazul. Pinii de forma An sunt pinii circuitului amplificator, pinii Mn sunt pinii multiplexorului, săgețile reprezintă ieșirile din circuitele logice, virgula reprezintă conexiuni multiple, caracterul ,,/” semnifică faptul că ieșirea depinde de semnalul de select al multiplexorului, iar liniuțele sunt conexiunile prin fire.

|  |  |
| --- | --- |
| **Funcționalitate pin** | **Legătura fizică** |
| Activare motor stânga | PTD3-A17->A3 |
| Activare motor dreapta | PTD2-A11->A9 |
| PWM motor stânga | PTA5-A6->A14-M10,M14->M12/M9 |
| PWM motor dreapta | PTD1-A4->A16-M2,M6->M4/M7 |
| PWM servomotor | PTA12-A8->A12 |
| Selecție direcție sens motoare | PTD5-A13->A7 |
| SI cameră | PTC6 |
| CLK cameră | PTC7 |
| AO cameră | PTB0 |
| AO senzor turație | PTB3 |
| DIN XBee | PTA1 |
| DOUT XBee | PTA2 |

Tabel 1 Configurație platformă amplicatoare

Outputul camerei este conectat la modulul ADC0 (singurul modul ADC existent pe FRDM-KL25Z) al platformei de dezvoltare în modul ,,single-ended” pe canalul 8 (ADC0\_SE8). Acest mod de conectare semnifică faptul că nu sunt nevoie două conexiuni pentru a măsura valoarea de pe pinul AO al camerei, modulul ADC folosind ca referință masa comună tuturor circuitelor și ca semnal de intrare o singura conexiune.

Dacă modulul LineScanCamera ar fi avut altă masă conectată decât cea a platformei și implicit a modulului ADC0, ar fi fost necesară o altă configurație al convertorului AD și anume, cea diferențială, fiind necesare două conexiuni fizice în loc de una pentru a putea măsura outputul. Un avantaj față de modul ,,single-ended” este eliminarea zgomotului prin torsadarea celor două conexiuni, însă, din observații, modul în care este folosit canalul modulului ADC0, nu introduce zgomot suficient de puternic încât să afecteze funcționalitatea camerei.

## **Platforma senzorilor de distanță**

Această plăcuță PCB are rolul de a fixa fizic și de a conecta senzorii de distanță la platforma mobilă. Au fost necesare trei PCB-uri, deoarece senzorii trebuie să măsoare distanțele pe trei direcții diferite. Acestea sunt fixate cu ajutorul unor tije metalice și lipite între ele pentru a forma o singură platformă. Conexiunile existente sunt vizibile în figura numărul 12, iar în figura numărul 13 este o fotografie cu platforma în realitate.



Fig. 12 Schemă platforma senzorilor de distanță



Fig. 13 Poză platforma senzorilor de distanță

Întrucât cei trei senzori vor începe măsurătorile în același timp, pinul Trig al fiecărui senzor este conectat la pinul de pe PCB-ul din mijloc de la poziția DH (Fig. 12). Acest lucru ușurează și implementarea software și ușurează și conectarea senzorilor la FRDM-KL25Z. Pinii de alimentare, VCC și GND sunt conectați la platforma amplificatoare la 3.3V și sunt comuni la cei trei senzori.

În următorul tabel sunt prezentate conexiunile dintre platforma de dezvoltare și platforma senzorilor sunt prezentate. Pinii de forma PTXn sunt pinii de pe plăcuța de dezvoltare, abrevierile de forma nYX reprezintă PCB-ul și coordonatele din figura numărul 12, iar liniuțele sunt conexiunile prin fire.

|  |  |
| --- | --- |
| **Funcționalitate pin** | **Legătura fizică** |
| Trig (toți senzorii) | PTD4-2CH |
| Echo senzor dreapta | PTD7-3AG |
| Echo senzor stânga | PTD6-2BM |
| Echo senzor mijloc | PTA17-1AF |

Tabel 2 Configurație platforma senzorilor de distanță

## **Conectarea tuturor platformelor**

Platforma de amplificare și platforma senzorilor de distanță sunt conectate la plăcuța de dezvoltare cu ajutorul unor jumperi, iar conexiunea dintre driverul de motoare și restul circuitelor de amplificare și multiplexare s-a realizat cu ajutorul unui cablu de tip ATA, cu 16 pini dispuși pe două rânduri, conectat la interfața JP2 aflată pe driverul de motoare (Fig. 11).

# **Implementarea software-ului**

În acest capitol vom descrie software-ul dezvoltat pentru realizarea cerințelor temei alese.

Proiectul constă într-o aplicație de tip ,,bare metal” dezvoltată cu ajutorul IDE-ului Keil µVision 5. ,,Bare metal” înseamnă că aplicația nu se folosește de un sistem de operare instalat deja pe platforma de dezvoltare și prin urmare, sistemul va fi mai rapid. O alternativă ar fi fost folosirea IDE-ului Mbed care ne pune la dispoziție sistemul de operare Mbed OS. Din considerente de viteză de procesare a sistemului s-a ales în final varianta ,,bare metal”. Acest lucru înseamnă, în schimb, că toate drivele pentru modulele folosite trebuie scrise de dezvoltator.

Proiectul este împărțit pe mai multe surse pentru a avea o lizibilitate mai mare în citirea codului.

## **Configurație generală**

IDE-ul folosit generează mai multe fișiere pentru a configura sistemul per ansamblu. Inițial, platforma este configurată să aibă frecvența procesorului de 20.97125MHz și frecvența magistralei tot 20.97125MHz. Pentru a mări viteza de procesare, s-a ales o altă configurație care setează procesorul să funcționeze la o frecvență de 48MHz și magistrala la 24MHz. Modulele TPM, UART, PIT folosesc toate ceasul MCGFLLCLK care are frecvența de 48MHz pentru această configurație.

## **Driver GPIO**

Modulul GPIO (General purpose input output) are rolul de a transmite sau de a primi date digitale sub forma unor biți (0 sau 1). Aproape toți pinii pot fi configurați să fie de acest tip. Exemple clasice de folosire a unor pini configurați astfel sunt: aprinderea/stingerea unui LED, activarea/dezactivarea unui modul, conectarea unor display-uri, etc.

Pentru a putea configura pinii să funcționeze ca GPIO aceștia trebuie să fie multiplexați corespunzător, configurând în registrul PORTx\_PCRn valoarea 1 în câmpul MUX, iar ceasul trebuie activat pentru portul corespunzător în registrul SIM\_SCGC5. Direcția pinului (adică dacă este de output sau input) se poate alege prin configurarea registrului GPIOx\_PDDR, citirea de la un pin se realizează cu ajutorul registrului GPIOx\_PDIR, iar setarea valorii de output se realizează cu ajutorul următoarelor registre: GPIOx\_PCOR (setează valoarea 0), GPIOx\_PSOR (setează valoarea 1), GPIOx\_PTOR (toggle).

La acest proiect s-au folosit cinci pini GPIO și anume: activare motor dreapta, activare motor stânga, direcție motoare, SI cameră și CLK cameră.

Pentru activarea motorului drept s-a folosit pinul PTD2. Acesta a fost configurat ca GPIO prin setarea valorii 1 în registrul PORTD\_PCR2 în câmpul MUX. Pentru ca pinul să fie de tip active high, au fost necesare încă două configurări în registrul PORTD\_PCR2 și anume, câmpul PE (valoarea 1) și câmpul PS (valoarea 0). Întrucât nu avem nevoie de întreruperi pentru această configurație, câmpul IRQC este setat pe valoarea 0. În final, pinul este configurat să fie output cu ajutorul registrului GPIOD\_PDDR și este setat la valoarea de logic high cu ajutorul registrului GPIOD\_PSOR. Pentru activarea motorului din stânga, se configurează similar pinul PTD3. Acești pini vor rămâne mereu pe valoarea de logic high întrucât nu avem nevoie să dezactivăm motoarele pentru acest proiect. Pinii pentru cameră și direcția motoarelor sunt și ei configurați similar.

Pentru a se alege sensul de mers înainte, pinul PTD5 trebuie să aibă valoarea logic low și invers pentru a schimba sensul de rotație.

## **Driver UART**

UART (Universal asyncronous reciever-transmiter) este un modul folosit la comunicarea serială dintre platforma de dezvoltare și un alt dispozitiv. Semnalul transmis este compus din doi biți, unul de start și unul de sfârșit și opt biți de date. Aceștia sunt transmiși la viteze prestabilite denumite baud rates care semnifică câți biți sunt transmiși pe secundă. Un exemplu de astfel de semnal putem observa în figura numărul 14.



Fig. 14 Semnal UART

În cazul nostru vom folosi modulul UART0 pentru a transmite date de la plăcuța de dezvoltare la modulul XBee. Pentru aceasta vom folosi pinii PTA1 și PTA2. Baud rate-ul folosit este de 9600 și se calculează în modul următor. Se alege o valoare de OSR (over sampling ratio) între 4 și 32 (valoarea implicită este 16), în cazul nostru 15, se adună cu unu și se înmulțește cu baud rate-ul dorit. Apoi se împarte frecvența ceasului folosit (în cazul acesta MCGFLLCLK) la valoarea calculată și rezultatul îl salvăm într-o variabila denumită sbr. Din această variabilă se scriu cei opt cei mai puțini semnificativi biți în registrul UART0\_BDL, iar biții de pe pozițiile 8-12 sunt memorați pe pozițiile 0-4 în registrul UART0\_BDH. Valoarea de OSR este memorată registrul UART0\_C4 în câmpul OSR. În continuare vom configura regiștrii UART0\_C1, UART0\_C3, UART0\_MA1, UART0\_MA2 și UART0\_C2 astfel:

* UART0\_C1 va avea toți biții setați pe 0;
  + Acest lucru ne asigură că mesajul are 8 biți și nu 9;
  + Generarea de bit de paritate este dezactivată;
  + Sunt selectați pini separați pentru RX și TX.
* UART\_C3 va avea biții setați pe 0;
  + Întrucât nu avem nevoie de întreruperi pentru UART, vor fi dezactivate;
* UART0\_MA1 și UART0\_MA2 vor avea toți biții setați pe 0;
  + Datele vor fi direct transmise către registrul UART0\_D.
* UART0\_C2 va avea câmpurile TE și RE setate pe valoarea 1, restul sunt lăsate la valoarea implicită.
  + TE și RE sunt biții de activare a transmițătorului respectiv al receptorului.

Pentru a transmite date se va scrie în registrul UART0\_D, iar pentru a citi date se va folosi același registru. Schemele logice ale transmițătorului și al receptorului sunt prezentate în următoarele două figuri.



Fig. 15 Schema transmițătorului UART



Fig. 16 Schema receptorului UART

Platforma autonomă ar putea funcționa și fără acest modul, întrucât acesta are doar rolul de debug. Testele realizate cu acest modul vor fi prezentate într-un capitol ulterior.

## **Driver TPM**

TPM este un numărător pe 16 biți care are următoarele funcționalități: generare de semnal PWM, input capture și output compare. Această placă de dezvoltare conține trei module TPM și anume: TPM0 care are șase canale disponibile, TPM1 are două canale disponibile și TPM2 care are tot două canale disponibile. Următoarea figură prezintă schema bloc a unui astfel de modul.



Fig. 17 Schema modulului TPM

Pentru numărare acest modul prezintă mai mulți parametri configurabili cum ar fi divizorul de ceas care poate lua valorile 1, 2, 4, 8, 16, 32, 64, 128, selectarea ceasului folosit, modul de numărare up counting sau up-down counting, valoarea până la care se numără și alții. Pentru acest proiect s-au folosit doar modurile input capture și generarea de pwm pentru controlul motoarelor, controlul servomotorului, măsurarea turației și generarea semnalulul TRIG pentru senzorii de distanță.

Modulul TPM0 este folosit pentru generarea de semnale PWM pentru motoare și pentru senzorii de distanță, TPM1 este folosit pentru sevomotor, iar TPM2 este folosit pentru senzorul de turație. Această alegere de a împărți modulele s-a datorat necesității de timpi diferiți pentru generarea flag-ului TOF și a numărului de canale disponibile pentru fiecare modul.

În continuare sunt prezentate două figuri care prezintă modurile de numărare up counting și up-down counting.



Fig. 18 TPM Up counting



Fig. 19 TPM Up-Down counting

* 1. **Generare PWM**

Pentru a genera semnalele PWM folosite la motoare și servomotor s-au folosit pinii PTA12 corespunzător modulului TPM1, canalul 0 și pinii PTA5 și PTD1 corespunzatori modulului TPM0, canalului 2, respectiv 1.

Pentru aceasta, a fost necesară configurarea acestor canale să genereze semnale PWM prin setarea valorilor ELSA, ELSB, MSA, MSB în registrul TPMx\_CnSC conform tabelului de mai jos pentru a alege Edge-aligned PWM, High-true pulses.



Tabel 3 TPMx\_CnSC Edge, level și mod de funcționare canal

Pentru a putea seta perioada semnalului ne vom folosi de registul TPMx\_MOD și vom seta valoarea 0x8CA0 și divizorul de ceas la valoarea 32 pentru TPM0 ceea ce corespunde la o perioada de aproximativ 40ms (valoare necesară pentru senzorii de distanță). Pentru a seta factorul de umplere, se va folosi registrul TPMx\_CnV care va fi setat dinamic în timpul execuției, cu excepția senzorilor de distanță, unde factorul de umplere va fi constant pentru semnalul TRIG, lațimea semnalului fiind de aproximativ 10us. Pentru servomotor vom seta valoarea din TPMx\_MOD la 0xEA60 și divizorul de ceas la valoarea 16 ceea ce corespunde la o perioada de aproximativ 20ms.



Fig. 20 TPM generare PWM

Pentru motoare s-a creat o funcție de setare a vitezei (denumită SetareViteza()) care primește ca parametru un număr de la zero la unu unde, zero înseamnă că motoarele sunt oprite (factorul de umplere 0% și implicit tensiunea de alimentare devine 0V) și unu înseamnă că motoarele funcționează la capacitate maximă. Totuși, pentru a nu accelera prea brusc motoarele s-a folosit o macrodefiniție care are valoarea 0.85 și este denumită MOTOARE\_VITEZA\_MAXIM\_SIG având rolul de a limita factorul de umplere care este setat în final.

Pentru sevomotoare s-a creat o funcție care să seteze unghiul roților (SetareUnghi()) și primește valori între -1 și 1, unde -1 reprezintă maxim stânga și 1 maxim dreapta. Întrucât roțile nu pot fizic să realizeze o rotație foarte mare, deoarece se lovesc de cadrul platformei, s-a creat și pentru această funcție o macrodefiniție denumită SERVOMOTOR\_MAXIM\_SIG care are valoarea de 0.4. Această macrodefiniție, ca și la motoare, a fost creată cu scopul de a calibra sistemul, întrucât funcțiile ar fi putut limita direct semnalele la aceste valori pe care le au macrodefinițiile.

* 1. **Input capture**

Acest mod de funcționare are rolul de a ne ajuta să numărăm câte fronturi crescătoare, descrescătoare sau ambele apar într-un anumit interval de timp. Pentru aceasta s-a folosit pinul PTB3 corespunzător modulului TPM2, canalul 1.

Pentru a configura acest canal să funcționeze în acest mod, se setează în registrul TPMx\_CnSC valorile ELSA, ELSB, MSA, MSB conform tabelului 3 astfel încât să fie ales modul Input capture, Capture on Rising Edge Only. Acest mod ne va genera o întrerupere la fiecare front crescător detectat.

TPM2 are registrul TPMx\_MOD setat la valoare 0xB71B și divizorul de ceas la valoarea 128 ceea ce corespunde la o perioada de aproximativ 125ms. Acest lucru înseamnă că vom putea măsura viteza de opt ori pe secundă. Pentru a realiza acest lucru, la fiecare front crescător, există o variabilă denumită nrInput care este incrementată. În momentul în care flag-ul TOF este setat și întreruperea a fost generată se calculează viteza curentă astfel:

(4.1)

Macrodefiniția NR\_INPUT\_COEFF este calculată în funcție de mai mulți parametrii statici și are următoarea formulă:

(4.2)

* PI, este constanta π și are valoarea 3.14;
* DIAMETER\_OF\_WHEEL, este diametrul roții exprimat în metri și are valoarea 0.05m;
* COEFFICIENT\_MEASURE\_TIME, este o constantă care ne va transforma din m/0.125s în m/s și are valoarea 0.125;
* NUMBER\_OF\_MAGNETS, este numărul de magneți de pe roată și are valoarea 6.

Se poate observa că toate valorile posibile sunt în funcție de NR\_INPUT\_COEFF și mai exact, multipli ai acestei constante (NR\_INPUT\_COEFF = 0.2093). Pentru a putea fi mai precisă măsurătoarea, ar trebui ca această constantă să aibă valoarea mai mică, iar singurul parametru care poate fi schimbat este perioada modulului TPM2. Dacă alegem o valoare mai mare a perioadei, răspunsul sistemului ar fi prea încet, dar precizia ar crește. Dacă s-ar alege o valoare mai mică a perioadei, pentru a avea un timp de răspuns mai bun, am avea în schimb următoarea problemă și anume: să presupunem că măsurăm odată la 10ms, asta ar însemna că pentru a fi siguri că măcar un magnet trece prin dreptul senzorului ar trebui ca roata să realizeze o șesime de revoluție în 10ms care este echivalentul în viteza la aproximativ 2.6m/s. Deci pentru a putea măsura viteza și să avem o valoare cât mai apropiată de realitate, platforma ar trebui să se deplaseze cu 2.6m/s și chiar și în acel caz, ar măsura doar 2.6m/s sau 5.2m/s în funcție de cum sunt plasați magneții la începutul unei măsurători. În cazul de față, cu perioada de 125ms, viteza minimă pentru a fi aproape siguri că detectăm un magnet este de aproximativ 0.20m/s ceea ce este mult mai aproape de vitezele pe care platforma le va atinge.

## **Driver PIT**

Modulul PIT este folosit pentru a măsura anumite perioade de timp cu ajutorul unor serii de numărătoare. Platforma FRDM-KL25Z are un singur modul PIT cu două canale. Aceste două canale sunt folosite pentru generarea semnalului CLK a modulului LineScanCamera și pentru generarea semnalului SI. Pentru aceasta, prima oară este activat modulul PIT cu ajutorul registrului PIT\_MCR prin setarea câmpului MDIS la valoarea 0. Apoi sunt activate întreruperile celor două canale cu ajutorul regiștrilor PIT\_TCTRLn prin setarea câmpului TIE la valoarea 1.

Pinul folosit pentru CLK este PTC7 și pentru SI este PTC6. Pentru a putea realiza o măsurătoare cu ajutorul camerei avem nevoie să generăm cele două semnale conform figurii prezentate pe următoarea pagină.



Fig. 21 Semnale LineScanCamera

Rolul canalului unu este de a genera semnalul CLK și de a începe o măsurătoare cu ajutorul modulului ADC. Acest lucru se realizează cu ajutorul întreruperii, unde valoarea pinului PTC7 este schimbată în zero dacă valoarea era unu și invers (toggle).

Rolul canalului zero este de a genera semnalul SI. Pentru aceasta, se vor considera patru stări:

* În prima stare se consideră că a fost terminată măsurătoarea anterioară și se pregătește următoarea. Se setează pinul SI pe valoarea de unu logic după care se setează canalul zero să numere timp de un sfert de perioadă a semnalului CLK;
* În a doua stare se setează CLK la valoarea de logic unu și se mai așteaptă încă un sfert de perioadă;
* În a treia stare, semnalul SI este setat la valoarea de logic zero și se așteaptă din nou un sfert de perioadă pentru starea finală;
* În ultima stare semnalul CLK este setat la valoarea de zero logic, canalul unu este activat și se începe o nouă măsurătoare.

## **Driver ADC**

Modulul ADC are rolul de a converti un semnal analogic într-un semnal digital. Placuța de dezvoltare conține un singur modul ADC și anume ADC0 care are patru perechi de canale diferențiale și 24 de canale ,,single-ended”. Structura acestui modul este prezentată în figura de mai jos.



Fig. 22 Schema modulului ADC

Convertoarele AD au mai multe moduri de a realiza conversia precum SAR și Delta-Sigma. În cazul nostru modulul ADC este de tip SAR și realizează conversia prin aproximări succesive, asemănător cu o căutare binară. Avantajul unei astfel de conversii este viteza cu care se realizează operațiile, dar se pierde din rezoluție, acest modul având o rezoluție maximă de 16 biți. O reprezentare simplificată a acestui mod de a realiza conversia este prezentată în următoarea figură cu o rezoluție de 4 biți.



Fig. 23 SAR pe 4 biți

Acest modul este folosit pentru a prelua output-ul de la modulul LineScanCamera. Pentru aceasta s-a folosit pinul PTB0 corespunzător canalului 8 ,,single-ended”. Registrul ADC0\_CFG1 a fost setat tot pe valoarea zero. Acest lucru ne asigură că modulul va realiza conversia cu o rezoluție de 8 biți, întrucât nu avem nevoie de o precizie foarte mare, sample time este setat să fie cât mai scurt, divizorul de ceas are valoarea unu, iar ceasul folosit pentru AD este Bus Clock. Registrul ADC0\_CFG2 este configurat astfel încât să folosit canalele ADxxa. Modulul este dezactivat pe timpul configurării prin scrierea valorii 0x1F în registrul ADC0\_SC1A. Întrucât vrem ca începerea unei conversii să se realizeze software și nu hardware se setează biții registrului ADC0\_SC2 pe zero. Acest lucru ne asigură și că DMA este dezactivat și funcția de comparare este dezactivată.

Înainte de a putea configura modulul acesta trebuie calibrat. Pentru aceasta este necesară selectarea modului în care modulul face o medie a 32 de sample-uri (nu este activată și pentru folosirea modulului propriu-zisă) și este setat bitul CAL. Se verifică momentul în care bitul COCO din registrul ADC0\_SC1A este setat moment în care se poate verifica dacă calibrarea a fost realizată cu succes sau nu. În final, trebuie setat gain-ul care se calculează conform manualului. Pașii care trebuie urmați sunt prezentați în continuare:

* Se inițializează o variabilă pe 16 biți.
* Se adună valorile pentru partea pozitivă a gain-ului din regiștrii ADC0\_CLP0, ADC0\_CLP1, ADC0\_CLP2, ADC0\_CLP3, ADC0\_CLP4 și ADC0\_CLPS și se memorează în variabila declarată la pasul anterior.
* Se împarte variabila la doi și se setează MSB-ul.
* Se salvează valoarea în registrul ADC0\_PG.
* Se repetă pașii pentru partea negativă a gain-ului.

Cu ajutorul acestui modul vom măsura fiecare pixel indiviual provenit de la pinul AO al modulului LineScanCamera. Aceștia vor fi salvați într-un vector care este folosit pentru debug. Pentru a detecta linia se calculează minimul acestor pixeli. Întrucât pe margini se observă că pixelii sunt mai întunecați și mulți ajung în afara traseului, minimul se va calcula de la CAMERA\_IGNORE\_EDGE\_VAL până la poziția 127-CAMERA\_IGNORE\_EDGE\_VAL, unde această macrodefiniție are valoarea 12, aleasă empiric. Capetele vectorului și pozițiile corespunzătoare marginilor create cu ajutorul macrodefinițiilor vor avea, doar pentru debug valori mari pentru a se vedea delimitarea.

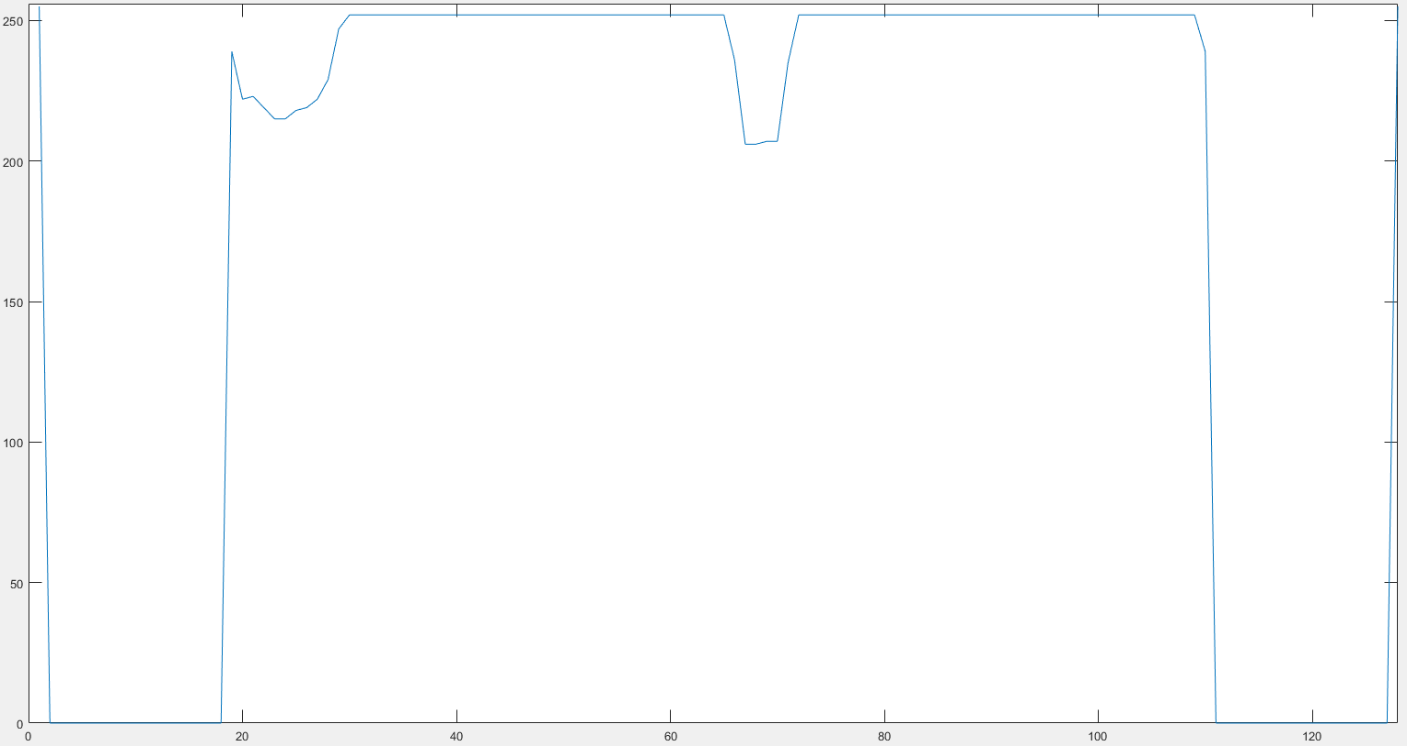


Fig. 24 Măsurătoare cameră

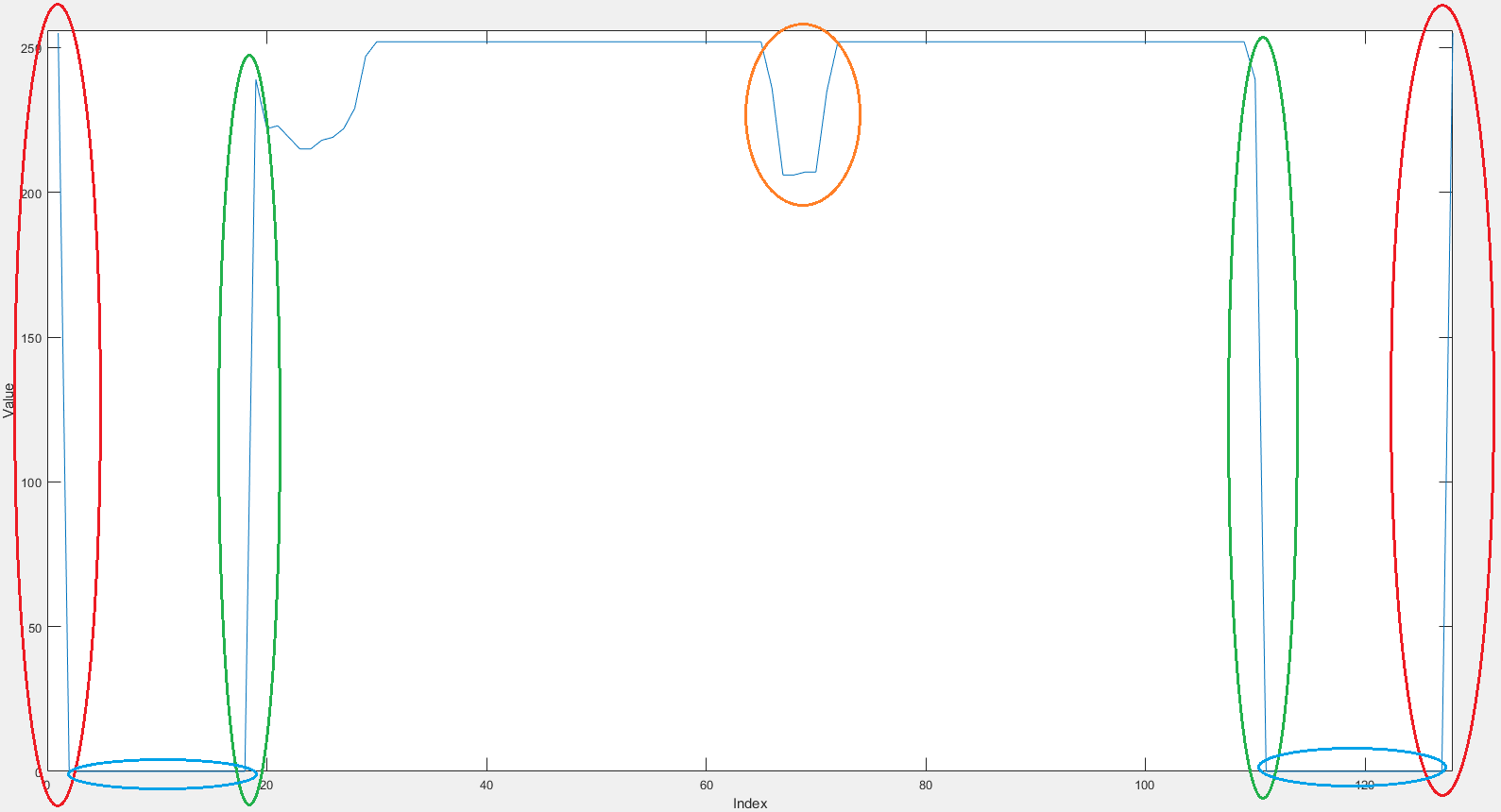


Fig. 25 Măsurătoare cameră cu indicații ale zonelor

* Roșu reprezintă marginile imaginii;
* Verde reprezintă marginile definite de CAMERA\_IGNORE\_EDGE\_VAL;
* Portocaliu este linia neagră;
* Albastru reprezintă zonele care nu sunt reținute la măsurare.

## **Menținerea drumului**

Detecția drumului se realizează cu ajutorul modulului LineScanCamera. Metoda este deja prezentată în subcapitolul Driver ADC. Pentru a decide direcția, s-a folosit funcția decideDirectiaDrumSimplu() din subsistemul Sistem Decizional. Această funcție primește ca parametru poziția liniei calculată în întreruperea modulului ADC și returnează o valoare între -1 și 1 pentru servomotor. Modul în care se calculează este astfel:

* Se inițializează o variabilă temp cu valoarea 128, adică numărul de pixeli dintr-o măsurătoare;
* Se scade de două ori SERVOMOTOR\_THRESHOLD din temp (macrodefiniția definește din ce punct motoarele vor vira maxim stânga, respectiv dreapta și are valoarea 12, aleasă empiric);
* Se face raportul dintre poziția liniei relativa la valoarea macrodefiniției SERVOMOTOR\_THRESHOLD și temp (Acest nou rezultat este o variabilă între 0 și 1);
* Se înmulțește cu doi rezultatul anterior și se scade unu.

Matematic calculul are următoarea formulă:

(4.3)

Pentru a se decide viteza, se folosește funcția decideVitezaDrumSimplu() din subsistemul Sistem Decizional. Această funcție primește ca parametru poziția liniei și returnează viteza dorită. Viteza este decisă în modul următor:

* Dacă linia se află în vecinătatea mijlocului (vecinătate delimitată de o macrodefiniție denumită LINE\_MAX\_ERROR\_FOR\_SPEED care are valoarea 3, aleasă empiric) atunci viteza este setată la viteza maximă ca pentru un drum drept;
* Dacă linia nu se află în vecinătatea mijlocului, viteza este setată la una mai mică pentru curbe.

Cele două viteze (pentru drum drept și pentru curbă) sunt definite de macrodefinițiile:

* MOTOARE\_VITEZA\_MAX\_MS;
* MOTOARE\_VITEZA\_CURBA\_MS;

(4.4)

(4.5)

* MAX\_VITEZA\_MULTIPLIER are valoarea 5;
* MIN\_VITEZA\_MULTIPLIER are valoarea 2;

## **PID**

PID este un controller proporțional, derivativ și integrativ care are scopul de a minimiza eroarea unui semnal de intrare. Schema generală a unui PID este prezentată în următoarea figură.



Fig. 26 Schemă PID

În cazul nostru, r(t) este viteza dorită, y(t) este viteza măsurată, u(t) este semnalul PWM al motoarelor, iar e(t) este eroarea. Întrucât avem un PID discret în cazul nostru, acesta este adaptat și are următoarea formulă:

(4.6)

* TS, este perioada de la o iterație la alta și are valoarea de 0.125s;
* KP, este coeficientul părții proporționale și are valoarea 0.35;
* KI, este coeficientul părții integrative și are valoarea 0.01;
* KD, este coeficientul părții derivative și are valoarea 0.0065.

Aceste valori au fost alese empiric după mai multe teste.

Fiecare coeficent afectează răspunsul sistemului într-un mod diferit. Componenta proporțională adaugă la răspuns o valoare proporțională cu eroarea. Acest lucru înseamnă că odată cu creșterea acestei componente, scade și timpul de răspuns al sistemului. Dezavantajul unui răspuns rapid este că poate duce la overshoot (ieșirea să fie prea mare) și poate duce la o eroare staționară. Alt dezavantaj este faptul că dacă coeficientul proporțional devine prea mare acest lucru poate duce la o instabilitate a sistemului.

Componenta integrativă are rolul de a micșora eroarea staționară. Această componentă calculează o sumă formată din erorile precedente înmulțite cu durata lor. Scopul este de a elimina eroarea staționară. Dezavantajul unei valori mari este că poate duce la un overshoot mare.

Componenta derivativă are rolul de a prezice comportamentul sistemului prin calcularea pantei dintre erori. Această componentă poate diminua overshootul și poate ajuta la stabilitatea sistemului când KD are valori mici, însă este foarte sensibilă la zgomot.

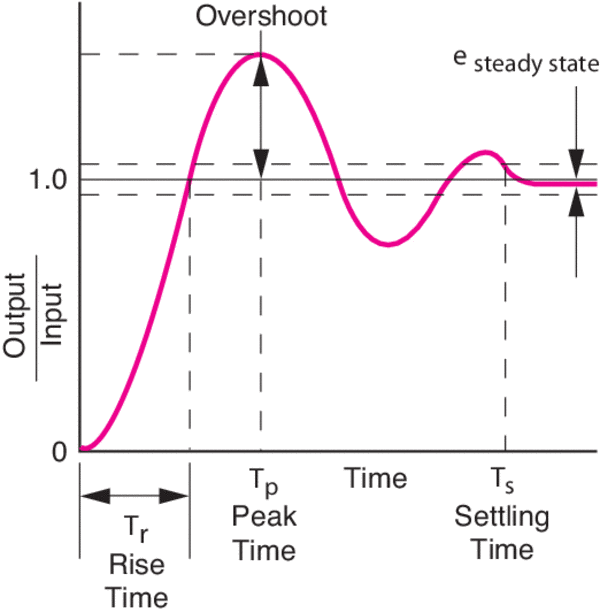


Fig. 27 PID, caracteristicile unui semnal

Pentru a seta factorului de umplere se calculează mai întâi turația, apoi viteza și în final folosind viteza actuală și cea cu o iterație în urmă, se calculează conform formulei u(n). Se verifică semnul acestei valori și se ia execută următorii pași:

* Dacă semnul valorii u(n-1) este diferit de u(n), se setează factorul de umplere al semnalelor PWM la zero și se schimbă sensul de rotație al motoarelor;
* Se apelează funcția SetareViteza() cu parametrul u(n) sau –u(n), dacă u(n) are valoare negativă;
* Se reține semnul valorii u(n) pentru următoarea iterație.

Întrucât viteza este măsurată în incremenți de NR\_INPUT\_COEFF și nu se vor observa valori intermediare, dacă viteza măsurată este apropiată de cea dorită se va considera eroarea zero.

## **Ocolirea obstacolelor**

Pentru a putea ocoli obstacolele s-au folosit trei senzori de distanță. Aceștia au nevoie de un semnal pentru a începe măsurătoarea, semnal care are perioada de 40ms și lățimea semnalului de 10us. Din momentul în care se începe măsurătoarea până în momentul în care se termină, semnalul ECHO va avea valoarea logică HIGH. În momentul în care semnalul ECHO are valoarea logică LOW, este activată o întrerupere (Pinul PTA17 pentru senzorul din mijloc, PTD7 pentru senzorul din dreapta, PTD6 pentru senzorul din stânga). Distanța se calculează conform următoarei formule:

(4.7)

Întrucât nu putem măsura timpul direct, vom calcula o constantă care să aibă unitatea de măsură cm/count unde count reprezintă valoarea din registrul TPM0\_CNT. Această constantă are următoarea formulă:

(4.8)

* 340, este viteza sunetului;
* 100, este o constantă pentru a transforma din metri în centimetrii;
* 1500000, este valoarea pe care ar trebui să aibă TPM0\_CNT pentru a avea perioada de o secundă;
* 2, este o constantă pentru a diviza distanța în două deoarece măsurătoarea include drumul dus-întors.

În final formula este rescrisă astfel:

(4.9)

* DISTANTA\_EROARE este o macrodefiniție necesară pentru a corecta măsurătoarea și are valoarea 8.1cm. Eroarea apare deoarece există un delay până la începerea măsurătorii, vizibil în figura de mai jos (galben este semnalul TRIG, albastru este semnalul ECHO).

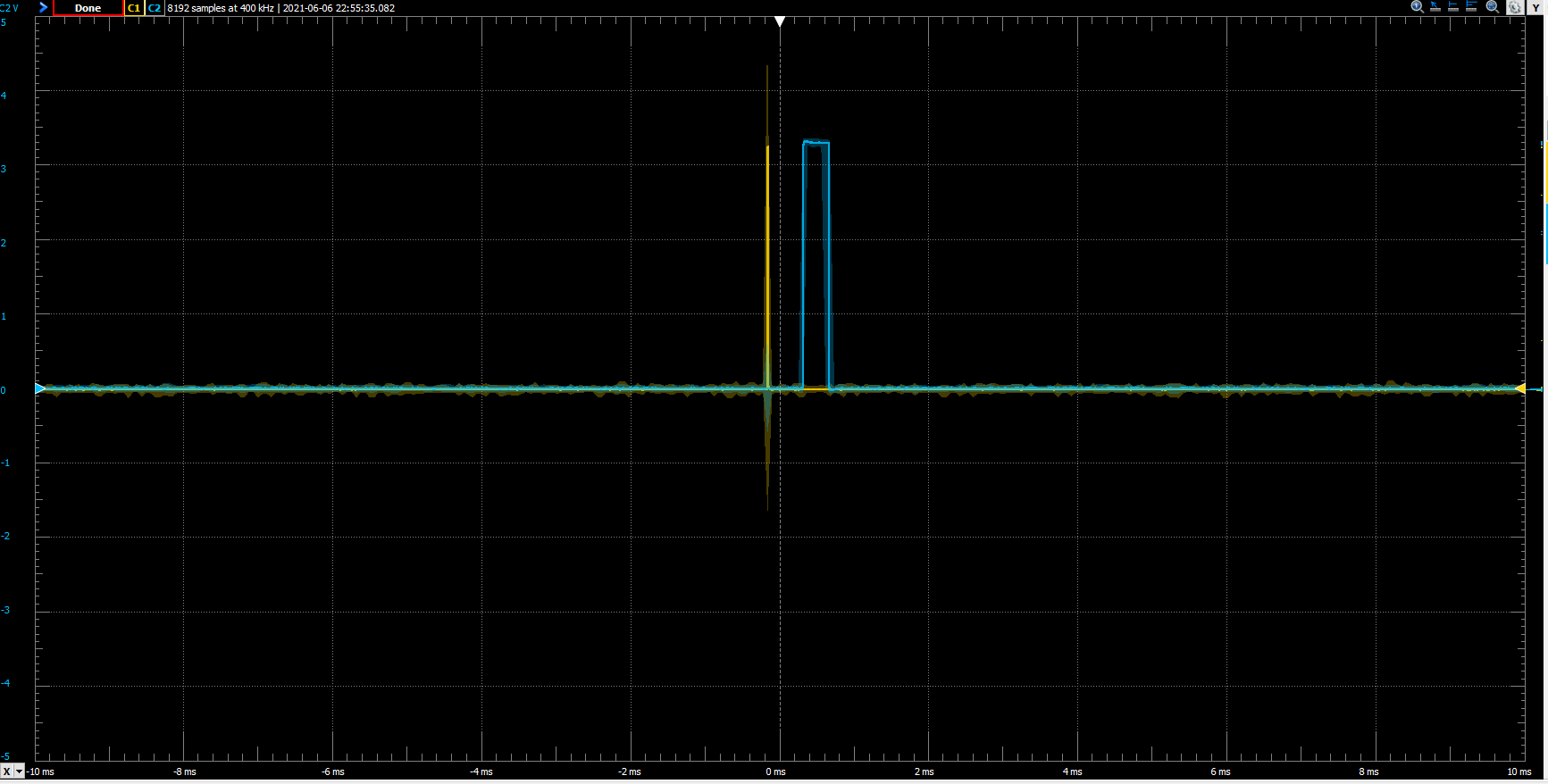


Fig. 28 Semnalele TRIG și ECHO

În momentul în care platforma măsoară o distanță mai mică decât DISTANTA\_THRESHOLD (macrodefiniție care are valoarea 30) pe oricare dintre direcțiile stânga sau drepta sau o distanță pe direcția din mijloc mai mică decât DISTANTA\_THRESHOLD\_MIJLOC (macrodefiniție care are valoarea 60), platforma va urma pașii:

* Se intră în altă stare unde linia va fi ignorată;
* Se reduce viteza la valoarea MOTOARE\_VITEZA\_OBSTACOL;

(4.9)

* MAX\_VITEZA\_OBS\_MULTIPLIER are valoarea 1.
* Se verfică dacă există obstacole pe ambele direcții stânga și dreapta;
* Dacă da, se virează în direcția opusă față de obstacolul mai apropiat;
* Dacă există doar pe una din cele două direcții se virează direct în direcția opusă;
* Dacă nu mai există niciun obstacol, se virează puțin către direcția unde s-a aflat ultimul obstacol detectat și se revine la starea inițială cu detecția liniei.

# **Testare**

## **Interacțiunea cu platforma**

Întrucât nu avem o interfață grafică, testarea nu s-a desfășurat foarte rapid, fiind necesare ajustări mici, recompilarea codului și repornirea platformei de fiecare data. Pot fi urmărite anumite date cu ajutorul modulului Xbee sau direct prin conexiune serială la un PC cu ajutorul unor scripturi scrise în Matlab.

## **Diagrame UML**

Modul în care utilizatorul interacționează cu platforma este reprezentat în diagrama de mai jos.

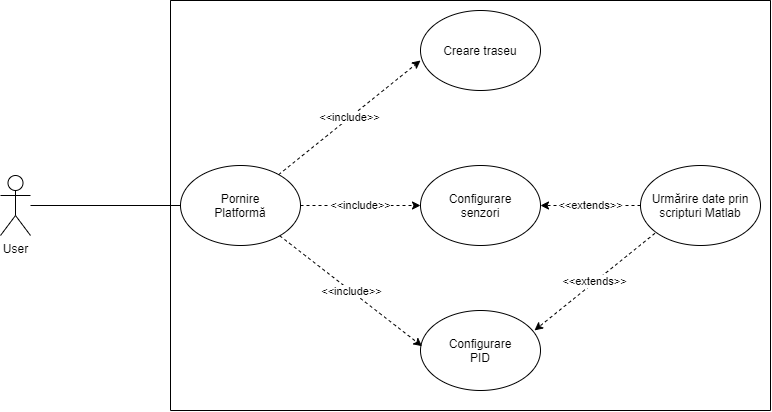


Fig. 29 Diagrama Use Case

În acest tip de diagrama, avem actorul, care în acest caz este utilizatorul platformei și următoarele stări:

* Pornire platformă, această stare reprezintă pornirea efectivă a platformei și plasarea acesteia pe traseu. Această stare nu poate fi de sine stătătoare având nevoie de stările Configurare senzori, Configurare PID și Creare Traseu;
* Creare Traseu, această stare reprezintă acțiunea fizică de a produce un traseu pe care platformă să îl parcurgă;
* Configurare senzori, această stare reprezintă configurarea parametrilor de threshold, a perioadelor pentru TPM, pentru a produce un comportament cât mai eficient și precis;
* Cpnfigurare PID, această stare reprezintă configurarea parametrilor KP, KD, KI, TS pentru pid pentru a putea duce la o accelerare și frânare cât mai rapidă și stabilă;
* Urmărire date prin scripturi de Matlab, această stare reprezintă urmărirea anumitor date trimise de către platformă cu ajutorul unei conexiuni seriale sau cu ajutorul modulelor XBee.

După ce utilizatorul a setat parametrii necesari, a construit traseul și a pornit alimentarea platformei, se vor inițializa toate driverele necesare, se vor inițializa variabilele corespunzător și din acest punct putem considera că platforma se va afla în una din două stări, cea în care platforma nu are niciun obstacol în cale (STATE\_DRUM\_FARA\_OBSTACOL) și cea în care a fost detectat un obstacol pe una din cele trei direcții (STATE\_DRUM\_OBSTACOL). Diferența majoră dintre cele două stări este că în prima este luată linia în considerare în timp ce în a doua sunt luate în considerare doar obstacolele. Comportamentul este descris mai în detaliu în diagrama de activități de pe următoarea pagină (figura 30).

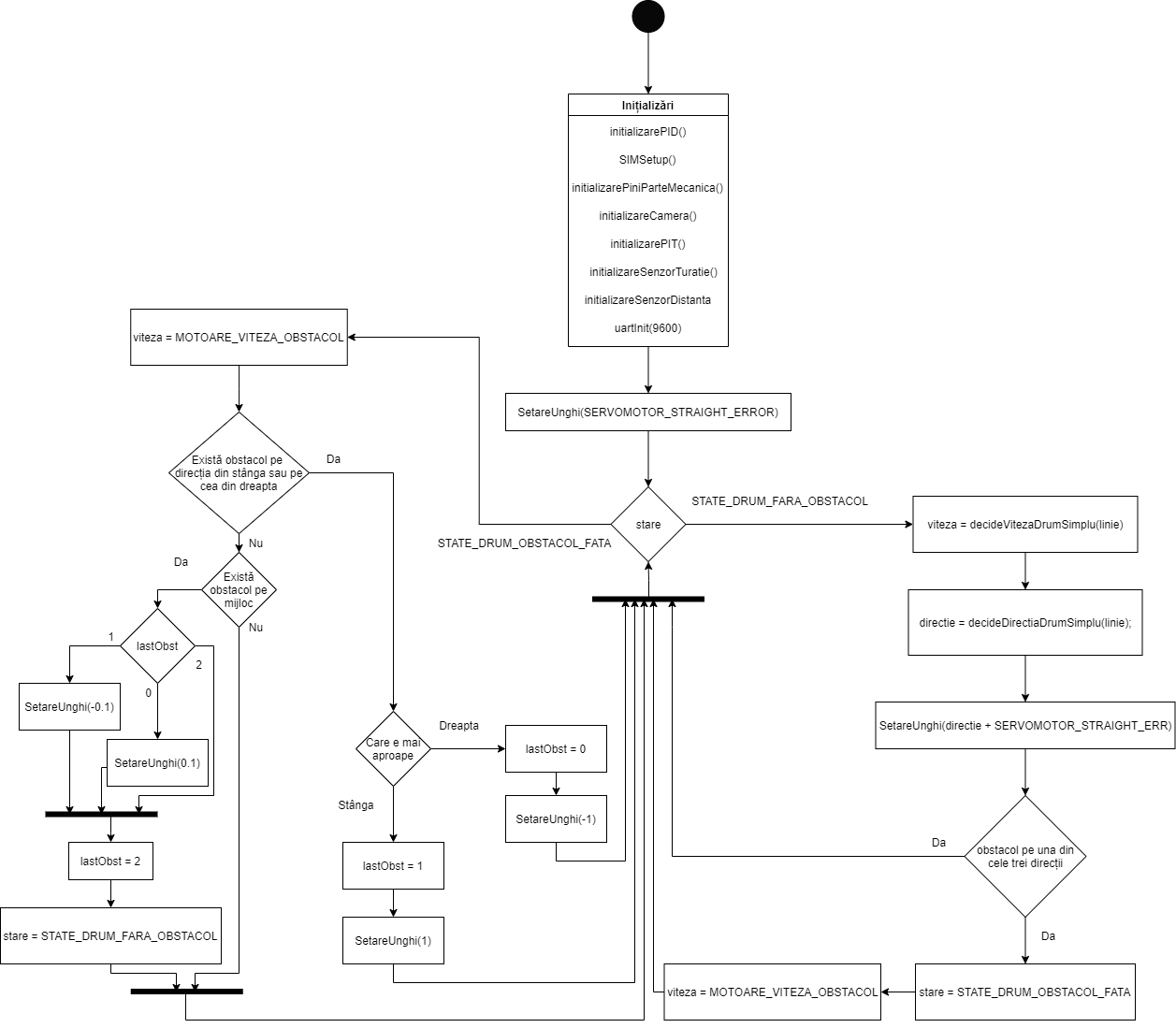


Fig. 30 Diagrama de activități

## **Testarea modulelor**

Pentru a se testa modulele s-au folosit mai multe metode:

* Transmiterea datelor prin interfața serială direct la un PC, după care au fost interpretate în Matlab;
  + Avantajul este că datele sunt transmise rapid și există risc foarte mic se a se pierde unele din ele;
  + Dezavantajul este că platforma trebuie să fie conectată fizic la PC, iar variabilele de tip float sunt rotunjite.
* Transmiterea datelor prin serială către modulul XBee și după cu ajutorul celui de-al doilea modul XBee, către PC și în final datele au fost interpretate de Matlab;
  + Avantajul este că datele sunt transmise fără a exista o conexiune fizică;
  + Dezavantajul este că datele au o întârziere vizibilă, se pierd unele dintre ele, variabilele de tip float sunt rotunjite și sunt necesare configurări suplimentare folosind utilitarul X-CTU.
* Folosirea debug-ului din Keil µVision;
  + Avantajul este că datele sunt exact cele din memorie, se pot vedea regiștrii ce valori au, iar variabilele de tip float nu sunt rotunjite;
  + Dezavantajul este că platforma trebuie conectată fizic la PC, iar în momentul în care se dorește vizualizarea unei varibile, software-ul trebuie oprit temporar.
* Măsurarea cu ajutorul osciloscopului;
  + Avantajul este că putem observa semnalul dorit așa cum arată în realitate, putem vedea forma, tensiunea, perioada și factorul de umplere;
  + Dezavantajul este că în unele locuri nu este practică folosirea acestuia, cum ar fi verificarea caracterelor transmise pe interfața UART, întrucât ar trebui să decodificăm fiecare mesaj transmis.
* Observarea efectelor în practică ale modificărilor parametrilor.
  + Avantajul este că putem realiza acest lucru rapid, doar alimentând platforma la tensiune și că putem vedea și practic efectul schimbării anumitor parametrii. În general se vor întâlni situații în care datorită mediul înconjurător (nivelul de luminozitate, contrastul dintre linie și fundal, etc.) platforma nu se va comporta conform așteptărilor și în acest caz, este cel mai bine să observăm practic ce se întâmplă în diferite condiții;
  + Dezavantajul este că nu putem observa valorile interne ale platformei prin această metoda și dacă sunt corecte.

De cele mai multe ori s-au folosit conexiunea serială, măsurarea cu ajutorul osciloscopului și observarea efectelor în practică.

Pentru cameră inițial s-a folosit osciloscopul. A fost creat un script în Matlab care citește de la seriala platformei 128 de valori după care actualizează în figura generată graficul astfel încât să avem o imagine live (vezi figura 24). Portul pe care se realizează comunicarea este de obicei COM7 (COM4 pentru XBee), iar baud rate-ul este 9600.

Pentru senzorul de turație și PID s-a folosit comunicarea prin XBee și s-a observat efectul modificărilor parametrilor la comportamentul practic al platformei. Viteza are o valoare reală ceea ce înseamnă că la transmiterea de date se va pierde din precizie. Pentru a contracara acest lucru vom trimite doar numărul de magneți prezenți și vom înmulți fiecare valoare primită în scriptul de Matlab cu 0.2093.

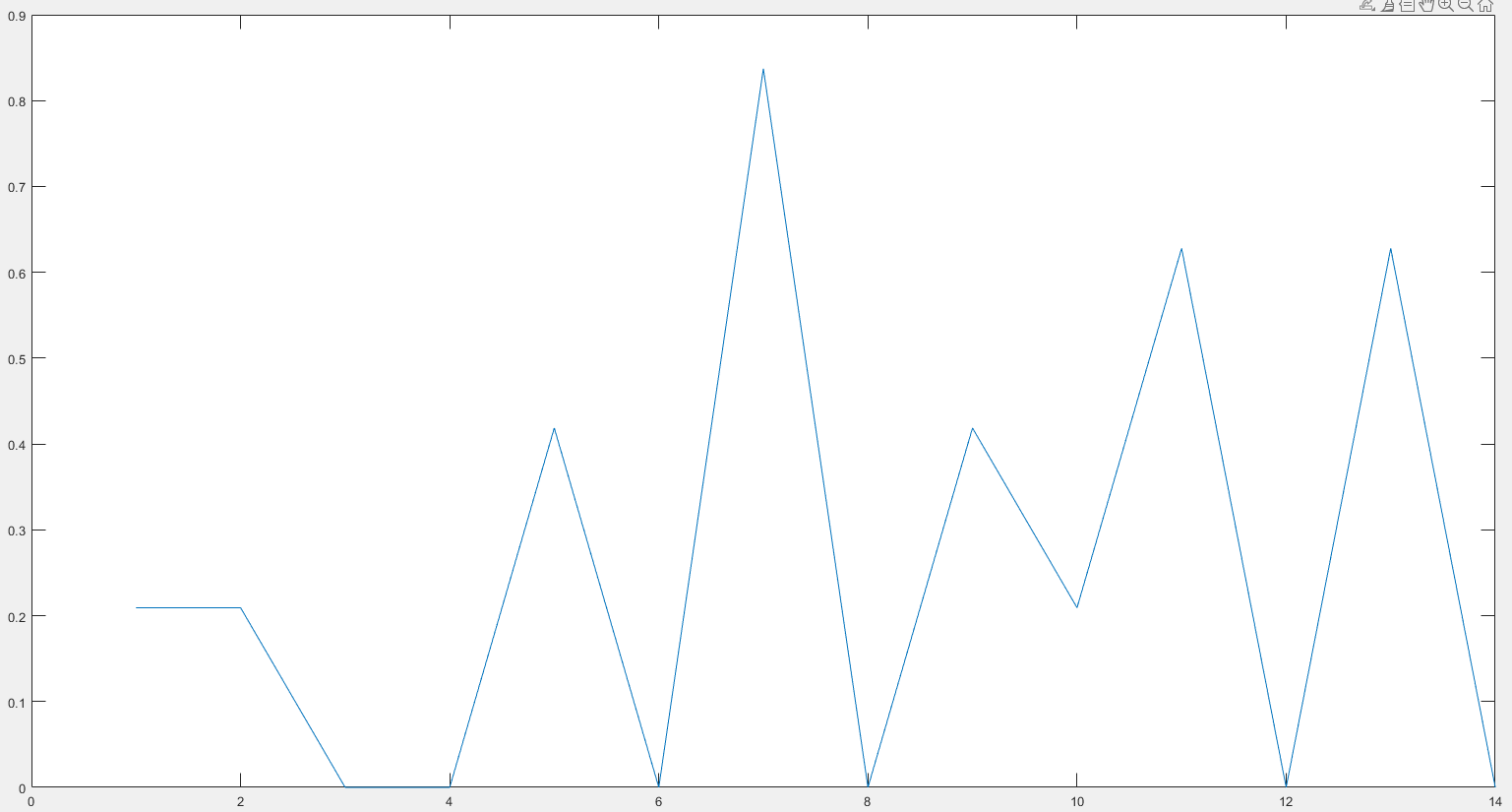


Fig. 31 Turația Matlab

În realitate platforma nu a avut niciodată, pe timpul acestei măsurători, viteza zero, dar datorită întârzierii adăugate de transmiterea de date și de probabilitatea de a nu nimeri un magnet la o măsurătoare și de-al nimeri la următoarea, graficul ajunge să aibă această formă. Pentru elimina din acest zgomot vom face o medie a punctelor apropiate. Acest semnal nou are o valoare mai apropiată de realitate.

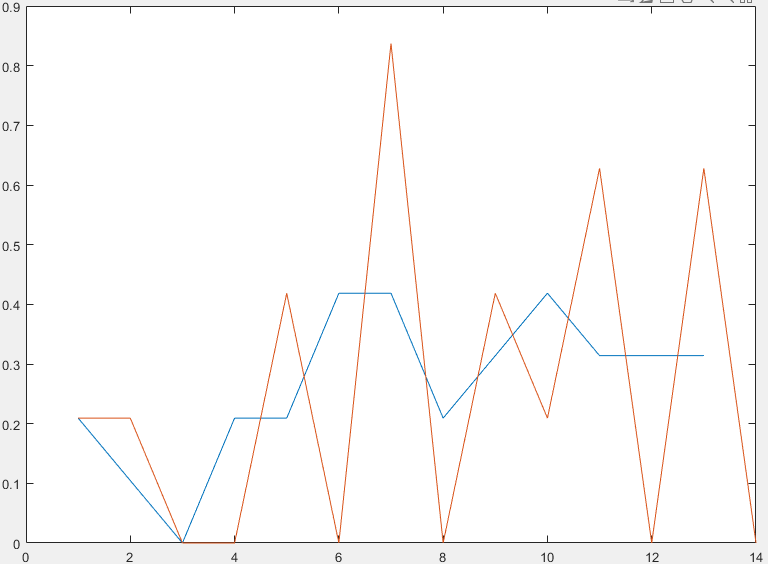


Fig. 32 Turația Matlab medie

În figura 32, semnalul cu culoarea portocalie este ceea ce a fost primit de la XBee, iar cel albastru este construit prin media punctelor apropiate.

Senzorii de distanță au fost testați cu ajutorul unui script în Matlab utilizând conexiunea serială a platformei și observați practic. Valorile obținute prin procesarea datelor de la senzorii de distanță sunt valori de tip float, însă nu avem nevoie de o precizie mai mică de un centimetru pentru această aplicație așa că datele sunt trimise direct pe interfața UART unde sunt transformate în numere naturale.

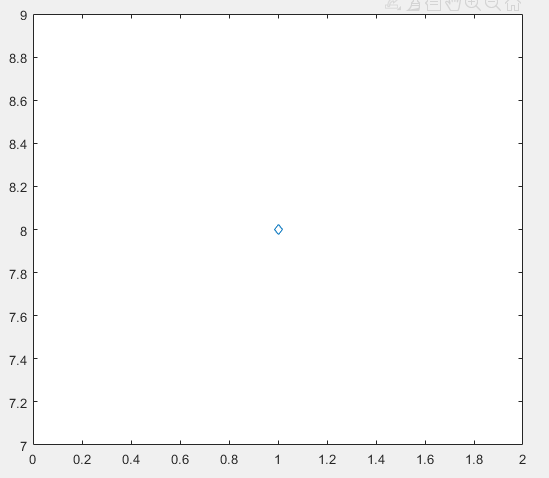


Fig. 33 Senzor distanță centru Matlab

În acest grafic se poate observa distanța față de senzorul de distanță de pe direcția centrală. Axa OX nu are nicio semnificație, iar pe axa OY la jumătate, în dreptul rombului, se află distanța față de un obstacol. Acest grafic se actualizează în timp real, schimbându-se doar valorile de pe OY.

# **Concluzii**

Această platformă de dezvoltare s-a potrivit perfect cerințelor proiectului. Aplicația ,,bare-metal” a fost o modalitate foarte bună de a acumula cunoștințe legate de platformele de dezvoltare din seria Kinetis și de a învăța cum funcționează aceastea.

## **Probleme întampinate**

Prima problemă întâmpinată a fost la configurarea pinilor. Inițial pinul PTA4 trebuia să fie folosit pentru canalul 1 al modulului TPM0 fiind mai aproape fizic de firul care duce la placa amplificatoare. Acest pin, însă, este preconfigurat și nu s-a reușit reconfigurarea lui ca pin pentru semnal PWM. În consecință a fost ales alt pin pentru a îndeplini această funcționalitate.

Următoarea problemă a fost întâmpinată la servomotor. Conexiunea fizică de la servomotor către roți se realizează cu ajutorul unor tije și a unui conector către o roată dințată care nu se potrivește perfect pentru a putea avea direcția înainte perfectă la un factor de umplere de 7.5%. Ca soluție la această problemă, în momentul în care se apelează funcția SetareUnghi() se adaugă, în general, un factor de corecție denumit SERVOMOTOR\_STRAIGHT\_ERR care este o macrodefiniție și are valoarea -0.07. De asemenea dacă platforma vira prea mult, roțile atingeau șasiul ceea ce ducea la o frânare a platformei. Această problemă a fost rezolvată limitând unghiul pe care funcția SetareUnghi() îl poate seta cu ajutorul macrodefiniției SERVOMOTOR\_MAX\_SIGNAL care are valoarea 0.4.

(6.1)

Alte două probleme întâmpinate sunt de natură mecanică. Modulul LineScanCamera este susținut fizic de patru șuruburi și două piulițe care datorita vibrațiilor provenite de la motoare, se desfac, iar cameră se rotește și nu mai este aliniată cu traseul. A doua problemă au fost magneții de pe roată, întrucât unul lipsea și măsurarea turației nu mai era precisă.

Pentru a ocoli obstacolele, senzorii de distanță transmit unde sonore care se reflectă asemănător cu undele electromagnetice. Acest lucru înseamnă că, ideal, ne dorim ca suprafața obstacolului să fie perpendiculară pe direcția de propagare a sunetului pentru a avea o probabilitate cât mai mare de a detecta reflexia. Întrucât platforma nu poate detecta obstacolul din anumite unghiuri, funcționalitatea de ocolire nu va funcționa mereu.

O problemă majoră a fost un scurtcircuit al servomotorului care ducea la supraîncălzirea siguranțelor termice F2 și F1, aflate pe driver-ul de motoare, încălzirea servomotorului și la un răspuns greu la semnalul de comandă sau chiar la niciun răspuns. Pentru a afla sursa problemei, s-a măsurat semnalul generat folosind un osciloscop. Acest lucru nu a dus la niciun rezultat și astfel, s-a înlocuit acesta cu totul moment în care problema supraîncălzirii siguranței F2 a fost rezolvată, problema siguranței termice F1 a fost ameliorată, iar servomotorul răspundea la comenzi. Siguranțele termice sunt prezentate în următoarele două figuri.

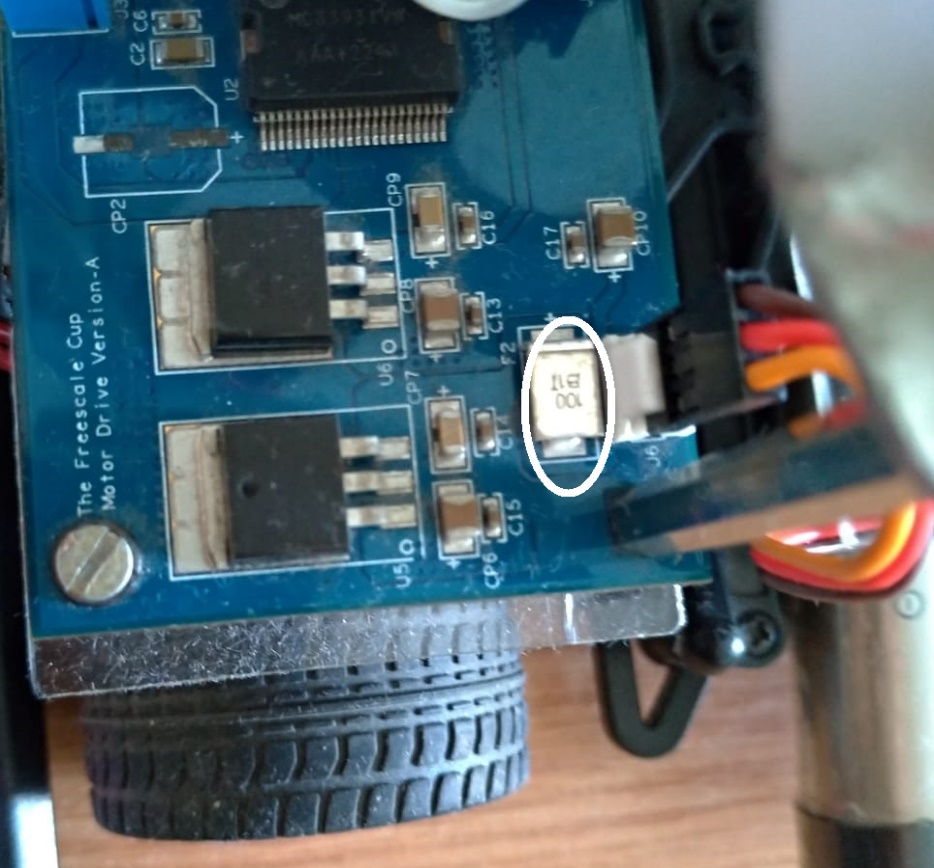


Fig. 34 Siguranța termică F2

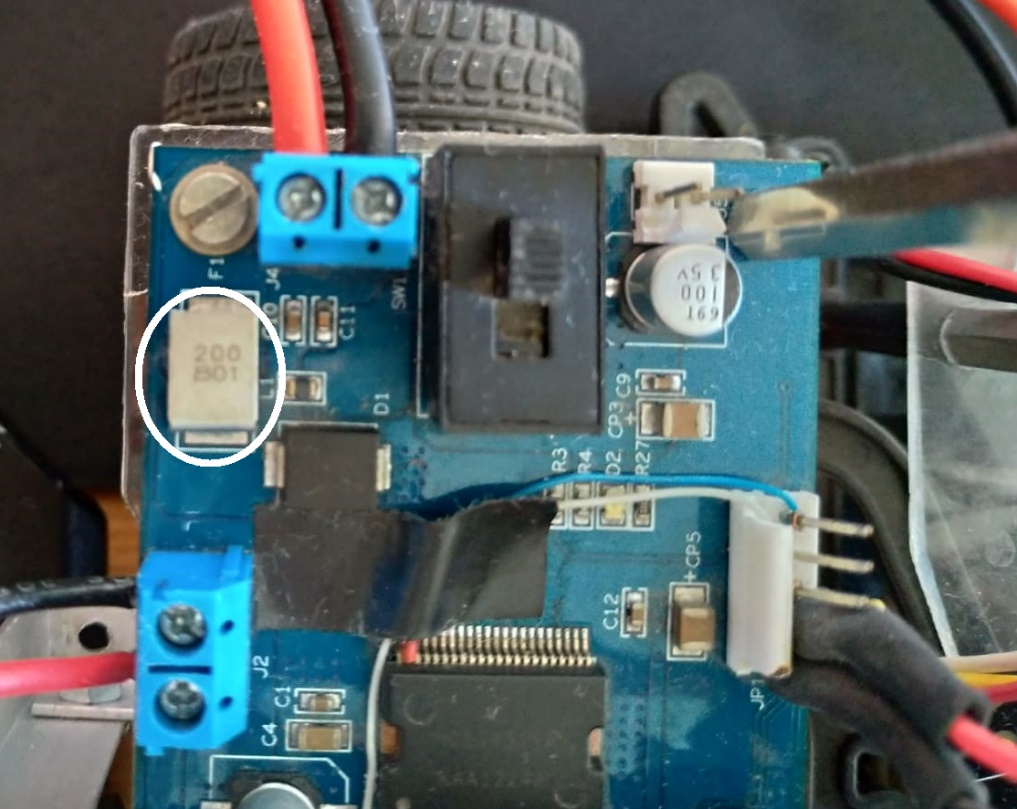


Fig. 35 Siguranța termică F1

Ultima problemă majoră a fost supraîncălzirea siguranței termice F1. Deși scurtcircuitul servomotorului a ajutat la încălzirea acesteia, nu a fost singura cauză. Puntea H responsabilă pentru motorul drept, încălzea prea puternic siguranța termică F1, ceea ce ducea la oprirea circuitului electric. Pentru a rezolva această problemă, deși avem semnale generate pentru două motoare, se va folosi doar puntea H responsabilă pentru motorul stâng.

# **Bibliografie**

Dean, G. (2017). *Embedded Systems Fundamentals with ARM Cortex-M based Microcontrollers, A Practical Aproach.*

Grandpierre, T. (n.d.). *Guide de de marrage et de de veloppement pour la NxP Cup.*

Guoxiaoli. (n.d.). *NXP community*. Retrieved from https://community.nxp.com/t5/University-Programs-Knowledge/Line-Scan-Camera-Use/ta-p/1105313#toc-hId-1303918968

NXP. (2012 Rev. 3). *KL25 Sub-Family Reference Manual.*

SparkFun. (n.d.). *SparkFun Learn*. Retrieved from https://learn.sparkfun.com/tutorials/xbee-wifi-hookup-guide/all

SparkFun. (n.d.). *Ultrasonic Ranging Module HC - SR04.*

# **Anexe**

## **Anexa 1**

Această anexă cuprinde fisierul Defines.h care conține toate macrodefinițiile folosite.

1. #include "MKL25Z4.h"
2. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Pini Senzori Distanta\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
3. #define PortSenzorDistantaTrig PORTD->PCR[4] //TPM0\_CH4
4. #define PortSenzorDistantaTrigMux 4
5. #define PortSenzorDistantaEchoS PORTD->PCR[6]
6. #define PortSenzorDistantaSPin 6
7. #define PortSenzorDistantaEchoSM 1
8. #define PortSenzorDistantaEchoD PORTD->PCR[7]
9. #define PortSenzorDistantaDPin 7
10. #define PortSenzorDistantaEchoDM 1
11. #define PortSenzorDistantaEchoC PORTA->PCR[17]
12. #define PortSenzorDistantaCPin 17
13. #define PortSenzorDistantaEchoCM 1
14. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Valori Senzor Distanta\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
15. #define SENZOR\_DISTANTA\_PWM 0xF
16. #define FACTOR\_MUL\_DISTANTA 0.01133f //cm/percount
17. #define DISTANTA\_EROARE 8.1f //cm
18. #define DISTANTA\_THRESHOLD 30 //cm
19. #define DISTANTA\_THRESHOLD\_MIJLOC 60 //cm
20. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Pini Senzor Turatie\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
21. #define PortSenzorTuratie PORTB->PCR[3] //TPM2\_CH1
22. #define PortSenzorTuratieMux 3
23. #define SenzorTuratieMOD 0xB71B
24. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*Valori Senzor Turatie\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
25. #define NUMBER\_OF\_MAGNETS 6
26. #define DIAMETER\_OF\_WHEEL 0.05f // metri
27. #define PI 3.14f
28. #define COEFFICIENT\_MEASURE\_TIME 0.125f
29. #define NR\_INPUT\_COEFF (PI \* DIAMETER\_OF\_WHEEL / COEFFICIENT\_MEASURE\_TIME) / NUMBER\_OF\_MAGNETS
30. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Pini Motor\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
31. #define PortMotorStanga PORTA->PCR[5] //TPM0\_CH2
32. #define PortMotorDreapta PORTD->PCR[1] //TPM0\_CH1
33. #define PortMotorActivareDreapta PORTD->PCR[2]
34. #define PortMotorActivareStanga PORTD->PCR[3]
35. #define PortSensMotor PORTD->PCR[5]
36. #define GPIODirMotorDreapta GPIOD\_PDDR
37. #define GPIODirMotorStanga GPIOD\_PDDR
38. #define GPIODirSensMotor GPIOD\_PDDR
39. #define GPIOSMotorDreapta GPIOD\_PSOR
40. #define GPIOSMotorStanga GPIOD\_PSOR
41. #define GPIOSSensMotor GPIOD\_PSOR
42. #define GPIOCMotorDreapta GPIOD\_PCOR
43. #define GPIOCMotorStanga GPIOD\_PCOR
44. #define GPIOCSensMotor GPIOD\_PCOR
45. #define GPIOPinMotorDreapta 3
46. #define GPIOPinMotorStanga 2
47. #define GPIOPinSensMotor 5
48. #define PortMotorStangaMux 3
49. #define PortMotorDreaptaMux 4
50. #define PortMotorActivareDreaptaM 1
51. #define PortMotorActivareStangaM 1
52. #define PortSensMotorMux 1
53. #define MotorMaxCount 0x8CA0
54. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Valori Motor\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
55. #define MOTOARE\_VITEZA\_MAXIM\_SIG 0.85f
56. #define SERVOMOTOR\_MAXIM\_SIG 0.4L
57. #define MOTOARE\_SENS\_INAITE 1
58. #define MOTOARE\_SENS\_SPATE -1
59. #define MAX\_VITEZA\_MULTIPLIER 5
60. #define MIN\_VITEZA\_MULTIPLIER 2
61. #define MAX\_VITEZA\_OBS\_MULTIPLIER 1
62. #define MOTOARE\_VITEZA\_MAX\_MS NR\_INPUT\_COEFF \* MAX\_VITEZA\_MULTIPLIER
63. #define MOTOARE\_VITEZA\_CURBA\_MS NR\_INPUT\_COEFF \* MIN\_VITEZA\_MULTIPLIER
64. #define MOTOARE\_VITEZA\_OBSTACOL NR\_INPUT\_COEFF \* MAX\_VITEZA\_OBS\_MULTIPLIER
65. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Pini ServoMotor\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
66. #define PortServoMotor PORTA->PCR[12] //TPM1\_CH0
67. #define PortServoMux 3
68. #define ServoMaxCount 0xEA60
69. #define ServoMinVal 0x5DC
70. #define ServoMaxVal 0x1D4C
71. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Valori Servomotor\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
72. #define SERVOMOTOR\_STRAIGHT\_ERR -0.07f
73. #define SERVOMOTOR\_MAX\_SIGNAL 0.4L
74. #define SERVOMOTOR\_THRESHOLD 12
75. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Valori TPM\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
76. #define TPM\_DIVIDER\_SERVO 4
77. #define TPM\_DIVIDER\_TURATIE 7
78. #define TPM\_DIVIDER\_MOTOARE 5
79. #define TPMClockSource 1
80. #define TPMPLLFLLSELValue 0
81. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Pini Camera\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
82. #define PortAOCamera PORTB->PCR[0] //ADC0\_SE8
83. #define PortAOCameraMux 0
84. #define PortSICamera PORTC->PCR[6]
85. #define PortSICameraMux 1
86. #define PortCLKCamera PORTC->PCR[7]
87. #define PortCLKCameraMux 1
88. #define GPIOPinSICamera 6
89. #define GPIOPinCLKCamera 7
90. #define GPIODirSICamera GPIOC\_PDDR
91. #define GPIODirCLKCamera GPIOC\_PDDR
92. #define GPIOSSICamera GPIOC\_PSOR
93. #define GPIOSCLKCamera GPIOC\_PSOR
94. #define GPIOCSICamera GPIOC\_PCOR
95. #define GPIOCCLKCamera GPIOC\_PCOR
96. #define GPIOTCLKCamera GPIOC\_PTOR
97. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Valori pentru camera\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
98. #define NUMBER\_OF\_CLOCKS 256
99. #define PIT\_HALF\_CLOCK 0x8FF
100. #define PIT\_QUARTER\_CLOCK PIT\_HALF\_CLOCK/2
101. #define PIT\_SI\_CLOCK PIT\_HALF\_CLOCK\*(256+8)
102. #define CAMERA\_START 0
103. #define CAMERA\_SET\_CLK 1
104. #define CAMERA\_CLEAR\_SI 2
105. #define CAMERA\_FINAL 3
106. #define CAMERA\_IGNORE\_EDGE\_VAL 12
107. #define LINE\_MAX\_ERROR\_FOR\_SPEED 3
108. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Pini ADC\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
109. #define ADCCameraResult ADC0\_RA
110. #define ADCCameraCFG1 ADC0\_CFG1
111. #define ADCCameraCFG2 ADC0\_CFG2
112. #define ADCCameraPG ADC0\_PG
113. #define ADCCameraMG ADC0\_MG
114. #define ADCCameraSC1A ADC0\_SC1A
115. #define ADCCameraSC1B ADC0\_SC1B
116. #define ADCCameraSC1 ADC0\_SC1
117. #define ADCCameraSC2 ADC0\_SC2
118. #define ADCCameraSC3 ADC0\_SC3
119. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PID\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
120. #define PID\_TS COEFFICIENT\_MEASURE\_TIME
121. #define PID\_KP 0.35f
122. #define PID\_KI 0.01f
123. #define PID\_KD 0.0065f
124. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Pini UART\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
125. #define PortUartRX PORTA->PCR[1]
126. #define PortUartTX PORTA->PCR[2]
127. #define PortUartRXMux 2
128. #define PortUartTXMux 2
129. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Valori UART\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
130. #define BUS\_CLOCK DEFAULT\_SYSTEM\_CLOCK
131. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Debug\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
132. #define CAMERA\_DEBUG\_LINE\_SCAN 0
133. #define SENZOR\_TUR\_SEMNAL\_DEBUG 0
134. #define SENZOR\_DISTANTA\_DEBUG\_S 0
135. #define SENZOR\_DISTANTA\_DEBUG\_D 0
136. #define SENZOR\_DISTANTA\_DEBUG\_C 0
137. #define DEZACTIVARE\_PID\_DEBUG 0
138. #define DEZACTIVARE\_MODUL\_DIST 0
139. #define CAMERA\_EDGE\_VAL 0xFF
140. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Stari\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
141. #define STATE\_DRUM\_FARA\_OBSTACOL 0
142. #define STATE\_DRUM\_OBSTACOL 1

## **Anexa 2**

Această anexă conține codul fișierului Main.c.

#include "MKL25Z4.h"

#include "Motoare.h"

#include "PortsSetup.h"

#include "Camera.h"

#include "SistemDecizional.h"

#include "SenzoriDistanta.h"

#include "Uart.h"

#include "PID.h"

extern uint8\_t linie**;**

extern float volatile viteza**;**

extern float volatile distantaS**;**

extern float volatile distantaD**;**

extern float volatile distantaC**;**

uint8\_t lastObst **=** 2**;**

static uint8\_t stare **=** STATE\_DRUM\_FARA\_OBSTACOL**;**

static long double directie**;**

int main **(**void**)** **{**

//Se initializeaza toate modulele si variabilele necesare

directie **=** 0**;**

SIMSetup**();**

initializarePID**(**PID\_TS**,** PID\_KP**,** PID\_KI**,** PID\_KD**);**

InitializarePiniParteMecanica**();**

initializareCamera**();**

initializarePIT**();**

initializareSenzorTuratie**();**

initializareSenzoriDistanta**();**

startCamera**();**

//9900 pentru XBEE, 115200 pentru UART prin cablu

uartInit**(**9600**);**

//Unghiul initial este de 0 grade

SetareUnghi**(**SERVOMOTOR\_STRAIGHT\_ERR**);**

**for(**int i**=**0**;**i**<**10000000**;**i**++){**

viteza **=** 0**;**

**}**

**while(**1**){**

**switch(**stare**)**

**{**

**case** STATE\_DRUM\_FARA\_OBSTACOL**:**

viteza **=** decideVitezaDrumSimplu**(**linie**);**

directie **=** decideDirectiaDrumSimplu**(**linie**);**

SetareUnghi**(**directie **+** SERVOMOTOR\_STRAIGHT\_ERR**);**

//Daca dorim sa nu folosim functionalitatea de ocolire de obstacole se poate seta macrodefinitia la valoarea 1

#if (DEZACTIVARE\_MODUL\_DIST == 0)

**if(**distantaC**<**DISTANTA\_THRESHOLD\_MIJLOC **||** distantaD **<** DISTANTA\_THRESHOLD **||** distantaS **<** DISTANTA\_THRESHOLD**)**

**{**

stare **=** STATE\_DRUM\_OBSTACOL**;**

viteza **=** 0**;**

**}**

#endif

**break;**

#if (DEZACTIVARE\_MODUL\_DIST == 0)

**case** STATE\_DRUM\_OBSTACOL**:**

viteza **=** MOTOARE\_VITEZA\_OBSTACOL**;**

//Daca avem obstacol pe laterale ne departam de cel mai apropiat

**if(**distantaD **<** DISTANTA\_THRESHOLD **||** distantaS **<** DISTANTA\_THRESHOLD**)**

**{**

**if(**distantaD **<** distantaS**)**

**{**

lastObst **=** 0**;**

SetareUnghi**(-**1**);**

**}**

**else**

**{**

lastObst **=** 1**;**

SetareUnghi**(**1**);**

**}**

**break;**

**}**

//Daca nu avem niciun obstacol pe nicio directie se revine la detectarea liniei si mentinerea drumului

**if(**distantaC **>** DISTANTA\_THRESHOLD\_MIJLOC**)**

**{**

**if(**lastObst **==** 0**)**

SetareUnghi**(**0.1f**);**

**if(**lastObst **==** 1**)**

SetareUnghi**(-**0.1f**);**

stare **=** STATE\_DRUM\_FARA\_OBSTACOL**;**

lastObst **=** 2**;**

**}**

**break;**

#endif

**default:**

**break;**

**}**

**}**

**}**

## **Anexa 3**

Această anexă conține codul fișierului Motoare.c.

#include "Motoare.h"

#include "Uart.h"

#include "PID.h"

float volatile vitezaCurenta **=** 0**;**

float volatile viteza**=**0**;**

float volatile semnal**=**0**;**

uint16\_t volatile nrInput **=** 0**;**

uint8\_t volatile lastSign **=** 0**;**

double abs**(**double x**)**

**{**

**if(**x**<**0**)**

**return** **-**x**;**

**return** x**;**

**}**

void TPM2\_IRQHandler**(**void**)**

**{**

NVIC\_ClearPendingIRQ**(**TPM2\_IRQn**);**

//Verificam daca intreruperea a fost generata de catre modul input capture

**if(**TPM2**->**CONTROLS**[**1**].**CnSC **&** TPM\_CnSC\_CHF\_MASK**)**

**{**

nrInput**++;**

TPM2**->**CONTROLS**[**1**].**CnSC **|=** TPM\_CnSC\_CHF\_MASK**;**

**}**

//Verificam daca modulul a terminat de numarat

**if((**TPM2\_SC **&** TPM\_SC\_TOF\_MASK**))**

**{**

//Calculam viteza

vitezaCurenta **=** nrInput**\***NR\_INPUT\_COEFF**;**

TPM2\_SC **|=** TPM\_SC\_TOF\_MASK**;**

//Daca PID-ul nu este dezactivat calculam semnalul urmator

**if(**DEZACTIVARE\_PID\_DEBUG **==** 0**)**

**{**

**if(**abs**(**viteza **-** vitezaCurenta**)<**NR\_INPUT\_COEFF**/**5**)**

**{**

vitezaCurenta **=** viteza**;**

**}**

//Daca masina trebuie sa se opreasca si viteza ajunge la 0 semnalul trebuie sa ramana pe 0

**if(**viteza **==** 0 **&&** vitezaCurenta **==** 0**)**

semnal **=** 0**;**

**else**

semnal **=** getNextPid**(**viteza**,** vitezaCurenta**);**

//PID-ul ne poate da valori mai mari decat 1 sau mai mici decat -1 si nu dorim acest lucru

**if(**semnal **>** 1**)**

semnal **=** 1**;**

**if(**semnal **<** **-**1**)**

semnal **=** **-**1**;**

//In functie de semnul variabilei semnal schimbam sensul motoarelor

**if(**semnal **<** 0**)**

**{**

**if(**lastSign **==** 0**)**

SetareViteza**(**0**);**

SetareSens**(**MOTOARE\_SENS\_SPATE**);**

SetareViteza**(-**semnal**);**

lastSign **=** 1**;**

**}**

**else**

**{**

**if(**lastSign **==** 1**)**

SetareViteza**(**0**);**

SetareSens**(**MOTOARE\_SENS\_INAITE**);**

SetareViteza**(**semnal**);**

lastSign **=** 0**;**

**}**

**}**

//Pentru a face debug pe semnal trimitem prin XBEE valoarea

#if (SENZOR\_TUR\_SEMNAL\_DEBUG == 1)

trimiteDate**(**nrInput**);**

#endif

//Resetam numarul de inputuri

nrInput **=** 0**;**

**}**

**}**

void InitializarePiniParteMecanica**(**void**){**

//Setare multiplexor pentru porturi

PortMotorDreapta **|=** PORT\_PCR\_MUX**(**PortMotorDreaptaMux**);**

PortMotorStanga **|=** PORT\_PCR\_MUX**(**PortMotorStangaMux**);**

PortMotorActivareDreapta **|=** PORT\_PCR\_MUX**(**PortMotorActivareDreaptaM**);**

PortMotorActivareStanga **|=** PORT\_PCR\_MUX**(**PortMotorActivareStangaM**);**

PortSensMotor **|=** PORT\_PCR\_MUX**(**PortSensMotorMux**);**

PortServoMotor **|=** PORT\_PCR\_MUX**(**PortServoMux**);**

//Configurare pini GPIO

PortMotorActivareDreapta **|=** PORT\_PCR\_PE**(**1**)** **&** **(~**PORT\_PCR\_PS**(**1**));**

PortMotorActivareStanga **|=** PORT\_PCR\_PE**(**1**)** **&** **(~**PORT\_PCR\_PS**(**1**));**

PortSensMotor **|=** PORT\_PCR\_PE**(**1**)** **&** **(~**PORT\_PCR\_PS**(**1**));**

GPIODirMotorDreapta **|=** 1**<<**GPIOPinMotorDreapta**;**

GPIODirMotorStanga **|=** 1**<<**GPIOPinMotorStanga**;**

GPIODirSensMotor **|=** 1**<<**GPIOPinSensMotor**;**

GPIOSMotorDreapta **|=** 1**<<**GPIOPinMotorDreapta**;**

GPIOSMotorStanga **|=** 1**<<**GPIOPinMotorStanga**;**

GPIOCSensMotor **|=** 1**<<**GPIOPinSensMotor**;**

//Setare divizor frecventa

TPM0**->**SC **|=** TPM\_SC\_PS**(**TPM\_DIVIDER\_MOTOARE**);**

TPM1**->**SC **|=** TPM\_SC\_PS**(**TPM\_DIVIDER\_SERVO**);**

//Setare mod de numarare

TPM0**->**SC **&=** **~**TPM\_SC\_CPWMS\_MASK**;**

TPM1**->**SC **&=** **~**TPM\_SC\_CPWMS\_MASK**;**

//Setare modulo

TPM0**->**CNT **=** 0**;**

TPM0**->**MOD **=** MotorMaxCount**;**

TPM1**->**CNT **=** 0**;**

TPM1**->**MOD **=** ServoMaxCount**;**

//Setare mod Edge-Aligned PMW

TPM1**->**CONTROLS**[**0**].**CnSC **=** TPM\_CnSC\_MSB\_MASK **|** TPM\_CnSC\_ELSB\_MASK**;**

TPM0**->**CONTROLS**[**2**].**CnSC **=** TPM\_CnSC\_MSB\_MASK **|** TPM\_CnSC\_ELSB\_MASK**;**

TPM0**->**CONTROLS**[**1**].**CnSC **=** TPM\_CnSC\_MSB\_MASK **|** TPM\_CnSC\_ELSB\_MASK**;**

//Setare mod trigger

TPM0**->**CONF **|=** TPM\_CONF\_TRGSEL**(**0x8**);**

TPM1**->**CONF **|=** TPM\_CONF\_TRGSEL**(**0x9**);**

//Setare factor de umplere

TPM1**->**CONTROLS**[**0**].**CnV **=** ServoMaxCount**\***0.075f**;**

TPM0**->**CONTROLS**[**2**].**CnV **=** 0**;**

TPM0**->**CONTROLS**[**1**].**CnV **=** 0**;**

//Activare TPM

TPM0**->**SC **|=** TPM\_SC\_CMOD**(**1**);**

TPM1**->**SC **|=** TPM\_SC\_CMOD**(**1**);**

**}**

void initializareSenzorTuratie**(**void**)**

**{**

PortSenzorTuratie **|=** PORT\_PCR\_MUX**(**PortSenzorTuratieMux**);**

PortSenzorTuratie **|=** PORT\_PCR\_PE**(**1**)** **&** **(~**PORT\_PCR\_PS**(**1**));**

//Setare divizor frecventa

TPM2**->**SC **|=** TPM\_SC\_PS**(**TPM\_DIVIDER\_TURATIE**);**

//Setare mod de numarare

TPM2**->**SC **&=** **~**TPM\_SC\_CPWMS\_MASK**;**

TPM2**->**SC **|=** TPM\_SC\_TOIE\_MASK**;**

//Setare modulo

TPM2**->**CNT **=** 0**;**

TPM2**->**MOD **=** SenzorTuratieMOD**;**

//Setare trigger

TPM2**->**CONF **|=** TPM\_CONF\_TRGSEL**(**0xA**);**

//Setare mod input capture

TPM2**->**CONTROLS**[**1**].**CnSC **=** TPM\_CnSC\_ELSB\_MASK **|** TPM\_CnSC\_CHIE\_MASK**;**

//Activare intrerupere

NVIC\_SetPriority**(**TPM2\_IRQn**,** 128**);**

NVIC\_ClearPendingIRQ**(**TPM2\_IRQn**);**

NVIC\_EnableIRQ**(**TPM2\_IRQn**);**

TPM2**->**SC **|=** TPM\_SC\_CMOD**(**1**);**

**}**

void SetareSens**(**int sens**)**

**{**

**if(**sens **==** MOTOARE\_SENS\_INAITE**)** // Setare sens inainte

**{**

GPIOCSensMotor **|=** 1**<<**GPIOPinSensMotor**;**

**}**

**else** **if(**sens **==** MOTOARE\_SENS\_SPATE**)** // Setare sens inapoi

**{**

GPIOSSensMotor **|=** 1**<<**GPIOPinSensMotor**;**

**}** //Daca nu este niciuna dintre ele nu se modifica sensul

**}**

void SetareViteza**(**float vitezaMotor**)**

**{**

//Daca viteza este in afara intervalului dorit se limiteaza

**if(**vitezaMotor **<** 0**)**

vitezaMotor **=** 0**;**

**if(**vitezaMotor **>** MOTOARE\_VITEZA\_MAXIM\_SIG**)**

vitezaMotor **=** MOTOARE\_VITEZA\_MAXIM\_SIG**;**

//Setam viteza corespunzator

TPM0**->**CONTROLS**[**2**].**CnV **=** MotorMaxCount**\***vitezaMotor**;**

TPM0**->**CONTROLS**[**1**].**CnV **=** MotorMaxCount**\***vitezaMotor**;**

**}**

void SetareUnghi**(**long double unghi**)**

**{**

//Intrucat la valoarea 0 platforma nu va mentine rotile fix drepte ne folosim de un factor de eroare

**if(**unghi **==** 0**)**

**{**

unghi **=** SERVOMOTOR\_STRAIGHT\_ERR**;**

**return;**

**}**

//Se limiteaza valorile posibile intrucat platforma nu poate vira la valori foarte mari sau mici

**if(**unghi**<-**SERVOMOTOR\_MAX\_SIGNAL**)**

**{**

unghi **=** **-**SERVOMOTOR\_MAX\_SIGNAL**;**

**}**

**if** **(**unghi **>** SERVOMOTOR\_MAX\_SIGNAL**)**

**{**

unghi **=** SERVOMOTOR\_MAX\_SIGNAL**;**

**}**

//Se calculeaza un procent

unghi**++;**

unghi **/=** 2**;**

unghi **=** **(**ServoMaxVal**-**ServoMinVal**)\***unghi**;**

unghi **=** unghi **+** ServoMinVal**;**

//Se seteaza valoarea calculata

TPM1**->**CONTROLS**[**0**].**CnV **=** unghi**;**

**}**

## **Anexa 4**

Această anexă conține codul fișierului PortsSetup.c.

#include "MKL25Z4.h"

#include "PortsSetup.h"

void SIMSetup**(**void**)**

**{**

//Setare biti in registrii SIM pentru ceasuri

SIM**->**SCGC5 **|=** SIM\_SCGC5\_PORTA\_MASK**;**

SIM**->**SCGC5 **|=** SIM\_SCGC5\_PORTB\_MASK**;**

SIM**->**SCGC5 **|=** SIM\_SCGC5\_PORTC\_MASK**;**

SIM**->**SCGC5 **|=** SIM\_SCGC5\_PORTD\_MASK**;**

SIM**->**SCGC5 **|=** SIM\_SCGC5\_PORTE\_MASK**;**

SIM**->**SCGC6 **|=** SIM\_SCGC6\_TPM0\_MASK**;**

SIM**->**SCGC6 **|=** SIM\_SCGC6\_TPM1\_MASK**;**

SIM**->**SCGC6 **|=** SIM\_SCGC6\_TPM2\_MASK**;**

SIM**->**SCGC6 **|=** SIM\_SCGC6\_ADC0\_MASK**;**

SIM**->**SCGC6 **|=** SIM\_SCGC6\_PIT\_MASK**;**

//Configurare ceas TPM

SIM**->**SOPT2 **|=** SIM\_SOPT2\_TPMSRC**(**TPMClockSource**);**

SIM**->**SOPT2 **&=~(**SIM\_SOPT2\_PLLFLLSEL**(**TPMPLLFLLSELValue**));**

**}**

## **Anexa 5**

Această anexă conține codul fișierului Camera.c.

#include "Camera.h"

#include "Uart.h"

#include "SistemDecizional.h"

#include "Motoare.h"

static volatile uint8\_t cameraState**=**0**;**

static volatile uint32\_t clockCycles**=**0**;**

static volatile uint8\_t min**;**

static volatile uint8\_t linieStatus **=** 63**;**

volatile uint8\_t linie**=**63**;**

#if (CAMERA\_DEBUG\_LINE\_SCAN == 1)

static volatile uint8\_t tempPixels**[**128**];**

static void debugLineScanCamera**(**void**)**

**{**

//Functie de debug pentru camera, se foloseste cu scriptul din Matlab

register uint8\_t i**=**0**;**

tempPixels**[**0**]** **=** CAMERA\_EDGE\_VAL**;**

tempPixels**[**127**]** **=** CAMERA\_EDGE\_VAL**;**

tempPixels**[**CAMERA\_IGNORE\_EDGE\_VAL**]** **=** CAMERA\_EDGE\_VAL**-**0x10**;**

tempPixels**[**127**-**CAMERA\_IGNORE\_EDGE\_VAL**]** **=** CAMERA\_EDGE\_VAL**-**0x10**;**

**for(**i**=**0**;**i**<**128**;**i**++)**

trimiteDate**(**tempPixels**[**i**]);**

**}**

#else

static void debugLineScanCamera**(**void**)**

**{**

**return;**

**}**

#endif

void ADC0\_IRQHandler**(**void**)**

**{**

uint8\_t value**;**

ADCCameraSC1A **&=** **~**ADC\_SC1\_AIEN\_MASK**;**

value **=(**uint8\_t**)**ADCCameraResult**;**

//Nu ne intereseaza marginea drumului intrucat ar putea sa fie din afara traseului si sa introduca erori

**if(**clockCycles **<** NUMBER\_OF\_CLOCKS **&&** clockCycles**/**2**>**CAMERA\_IGNORE\_EDGE\_VAL **&&** clockCycles**/**2**<**127**-**CAMERA\_IGNORE\_EDGE\_VAL**)**

**{**

#if (CAMERA\_DEBUG\_LINE\_SCAN == 1)

tempPixels**[**clockCycles**/**2**]** **=** **(**uint8\_t**)**value**;**

#endif

**if(**min**<(**uint8\_t**)**value**)**

**{**

linieStatus **=** clockCycles**/**2**;**

min **=** value**;**

**}**

**}**

**}**

void PIT\_IRQHandler**(**void**)**

**{**

NVIC\_ClearPendingIRQ**(**PIT\_IRQn**);**

**if(**PIT**->**CHANNEL**[**0**].**TFLG **&** PIT\_TFLG\_TIF\_MASK**)**

**{**

PIT\_TCTRL0 **=** 0**;**

PIT\_TCTRL1 **=** 0**;**

//Functie pentru a lua datele de la camera, aceasta are mai multe stari in functie de ce semnale se genereaza

**switch(**cameraState**)**

**{**

**case** CAMERA\_START**:**

GPIOCCLKCamera **=** 1**<<**GPIOPinCLKCamera**;**

debugLineScanCamera**();**

linie **=** linieStatus**;**

GPIOSSICamera **=** 1**<<**GPIOPinSICamera**;**

clockCycles **=** 1**;**

cameraState **=** CAMERA\_SET\_CLK**;**

PIT**->**CHANNEL**[**0**].**LDVAL **=** PIT\_QUARTER\_CLOCK**;**

min **=** 0x0**;**

**break;**

**case** CAMERA\_SET\_CLK**:**

GPIOSCLKCamera **|=** 1**<<**GPIOPinCLKCamera**;**

cameraState **=** CAMERA\_CLEAR\_SI**;**

**break;**

**case** CAMERA\_CLEAR\_SI**:**

GPIOCSICamera **|=** 1**<<**GPIOPinSICamera**;**

cameraState **=** CAMERA\_FINAL**;**

**break;**

**case** CAMERA\_FINAL**:**

GPIOCCLKCamera **|=** 1**<<**GPIOPinCLKCamera**;**

PIT**->**CHANNEL**[**0**].**LDVAL **=** PIT\_SI\_CLOCK**;**

ADCCameraSC1A **=** ADC\_SC1\_ADCH**(**8**)** **|** ADC\_SC1\_AIEN\_MASK**;**

PIT**->**CHANNEL**[**1**].**TCTRL **=** PIT\_TCTRL\_TEN\_MASK **|** PIT\_TCTRL\_TIE\_MASK**;**

cameraState **=** CAMERA\_START**;**

**break;**

**}**

PIT**->**CHANNEL**[**0**].**TFLG **&=** PIT\_TFLG\_TIF\_MASK**;**

PIT**->**CHANNEL**[**1**].**TFLG **&=** PIT\_TFLG\_TIF\_MASK**;**

PIT**->**CHANNEL**[**0**].**TCTRL **=** PIT\_TCTRL\_TEN\_MASK **|** PIT\_TCTRL\_TIE\_MASK**;**

**}**

**else** **if(**PIT**->**CHANNEL**[**1**].**TFLG **&** PIT\_TFLG\_TIF\_MASK**)**

**{**

//Aici se genereaza clockul pentru camera

**if(**clockCycles**%**2 **==** 0**)**

ADCCameraSC1A **=** ADC\_SC1\_ADCH**(**8**)** **|** ADC\_SC1\_AIEN\_MASK**;**

PIT**->**CHANNEL**[**1**].**TFLG **&=** PIT\_TFLG\_TIF\_MASK**;**

GPIOTCLKCamera **|=** 1**<<**GPIOPinCLKCamera**;**

clockCycles**++;**

**}**

**}**

void startCamera**(**void**)**

**{**

//Se porneste modulul PIT pentru camera

PIT\_MCR **&=** **~(**PIT\_MCR\_MDIS\_MASK**);**

PIT\_LDVAL0 **=** PIT\_QUARTER\_CLOCK**;**

PIT\_LDVAL1 **=** PIT\_HALF\_CLOCK**;**

PIT\_TCTRL0 **=** PIT\_TCTRL\_TEN\_MASK **|** PIT\_TCTRL\_TIE\_MASK**;**

PIT\_MCR **=** 1**;**

cameraState **=** CAMERA\_START**;**

**}**

void initializarePIT**(**void**)**

**{**

//Se activeaza modulul

PIT**->**MCR **&=** **~**PIT\_MCR\_MDIS\_MASK**;**

//Se configureaza canalul 0

PIT**->**CHANNEL**[**0**].**LDVAL **=** PIT\_LDVAL\_TSV**(**PIT\_HALF\_CLOCK**);**

PIT**->**CHANNEL**[**0**].**TCTRL **&=** PIT\_TCTRL\_CHN\_MASK**;**

PIT**->**CHANNEL**[**0**].**TCTRL **|=** PIT\_TCTRL\_TIE\_MASK**;**

//Se seteaza intrerupea

NVIC\_SetPriority**(**PIT\_IRQn**,** 128**);**

NVIC\_ClearPendingIRQ**(**PIT\_IRQn**);**

NVIC\_EnableIRQ**(**PIT\_IRQn**);**

**}**

void initializareCamera**(**void**)**

**{**

//Configurare pini CLK si SI

PortSICamera **|=** PORT\_PCR\_MUX**(**PortSICameraMux**);**

PortCLKCamera **|=** PORT\_PCR\_MUX**(**PortCLKCameraMux**);**

PortSICamera **|=** PORT\_PCR\_PE**(**1**)** **&** **(~**PORT\_PCR\_PS**(**1**));**

PortCLKCamera **|=** PORT\_PCR\_PE**(**1**)** **&** **(~**PORT\_PCR\_PS**(**1**));**

//Configurare pini GPIO

GPIODirSICamera **|=** 1**<<**GPIOPinSICamera**;**

GPIODirCLKCamera **|=** 1**<<**GPIOPinCLKCamera**;**

GPIOSCLKCamera **|=** 1**<<**GPIOPinCLKCamera**;**

GPIOCSICamera **|=** 1**<<**GPIOPinSICamera**;**

//Configurare pin pentru AO

PortAOCamera **|=** PORT\_PCR\_MUX**(**PortAOCameraMux**);**

//Se verifica calibrarea

**if(**calibrareADC**()==-**1**)**

**return;**

ADCCameraCFG1 **=** 0**;**

ADCCameraCFG2 **=** 0**;**

ADCCameraSC1A **=** 0x1F**;**

ADCCameraSC2 **=** 0**;**

ADCCameraSC3 **&=** 0xF**;**

NVIC\_SetPriority**(**ADC0\_IRQn**,** 128**);**

NVIC\_ClearPendingIRQ**(**ADC0\_IRQn**);**

NVIC\_EnableIRQ**(**ADC0\_IRQn**);**

**}**

static int calibrareADC**(**void**)**

**{**

//Functie de calibrare a ADC-ului, pentru detalii vedeti in reference manual

uint16\_t calVar**;**

ADCCameraSC2 **&=** **~**ADC\_SC2\_ADTRG\_MASK**;**

ADCCameraSC3 **&=** **(~**ADC\_SC3\_ADCO\_MASK **&** **~**ADC\_SC3\_AVGS\_MASK**);**

ADCCameraSC3 **|=** ADC\_SC3\_AVGE\_MASK **|** ADC\_SC3\_AVGS**(**3**);**

ADCCameraSC3 **|=** ADC\_SC3\_CAL\_MASK**;**

**while(!(**ADCCameraSC1A **&** ADC\_SC1\_COCO\_MASK**));**

**if(**ADCCameraSC3 **&** ADC\_SC3\_CALF\_MASK**)**

**return** **-**1**;**

calVar **=** 0**;**

calVar **+=**ADC0\_CLP0**;**

calVar **+=**ADC0\_CLP1**;**

calVar **+=**ADC0\_CLP2**;**

calVar **+=**ADC0\_CLP3**;**

calVar **+=**ADC0\_CLP4**;**

calVar **+=**ADC0\_CLPS**;**

calVar **/=** 2**;**

calVar **|=** 0x8000**;**

ADCCameraPG **=** ADC\_PG\_PG**(**calVar**);**

calVar **=** 0**;**

calVar **+=** ADC0\_CLM0**;**

calVar **+=** ADC0\_CLM1**;**

calVar **+=** ADC0\_CLM2**;**

calVar **+=** ADC0\_CLM3**;**

calVar **+=** ADC0\_CLM4**;**

calVar **+=** ADC0\_CLMS**;**

calVar **/=** 2**;**

calVar **|=** 0x8000**;**

ADCCameraMG **=** ADC\_MG\_MG**(**calVar**);**

ADCCameraSC3 **&=** **~(**ADC\_SC3\_CAL\_MASK**);**

**return** 0**;**

**}**

## **Anexa 6**

Această anexă conține codul fișierului SistemDecizional.c.

#include "SistemDecizional.h"

inline long double decideDirectiaDrumSimplu**(**uint8\_t linie**)** //Se va modifica pentru obstacole si curbe

**{**

//In functie de linie se calculeaza o valoare pentru unghi intre -1 si 1

long double temp **=** 128**;**

temp**-=**2**\***SERVOMOTOR\_THRESHOLD**;**

temp**=(**linie **-** SERVOMOTOR\_THRESHOLD**)/**temp**;**

temp**\*=**2**;**

temp**-=**1**;**

**return** temp**;**

**}**

inline float decideVitezaDrumSimplu**(**uint8\_t linie**)**

**{**

//In functie de pozitia liniei, platforma se va deplasa mai repede sau mai incet

**if(**linie**>=**63 **+** LINE\_MAX\_ERROR\_FOR\_SPEED **||** linie**<=**64 **-** LINE\_MAX\_ERROR\_FOR\_SPEED**)**

**{**

**return** MOTOARE\_VITEZA\_CURBA\_MS**;**

**}**

**else**

**{**

**return** MOTOARE\_VITEZA\_MAX\_MS**;**

**}**

**}**

## **Anexa 7**

Această anexă conține codul fișierului Uart.c.

#include "Uart.h"

uint32\_t osr **=** 15**;**

uint16\_t sbr**;**

uint8\_t temp**;**

extern uint8\_t cameraPixels**[**128**];**

void uartInit**(**uint32\_t baud**){**

//Activam ceasul pentru modulul UART0

SIM**->**SCGC4 **|=** SIM\_SCGC4\_UART0\_MASK**;**

//Dezactivam RX, TX pentru configurare

UART0**->**C2 **&=** **~**UART0\_C2\_TE\_MASK **&** **~**UART0\_C2\_RE\_MASK**;**

//Setam ceasul

SIM**->**SOPT2 **|=** SIM\_SOPT2\_UART0SRC**(**01**);**

SIM**->**SOPT2 **|=** SIM\_SOPT2\_PLLFLLSEL\_MASK**;**

//Setam pinii folositi

PORTA**->**PCR**[**1**]** **=** PORT\_PCR\_ISF\_MASK **|** PORT\_PCR\_MUX**(**2**);**

PORTA**->**PCR**[**2**]** **=** PORT\_PCR\_ISF\_MASK **|** PORT\_PCR\_MUX**(**2**);**

sbr **=** **(**uint16\_t**)((**BUS\_CLOCK**)/(**baud **\*** **(**osr**+**1**)));**

temp **=** UART0**->**BDH **&** **~(**UART0\_BDH\_SBR**(**0x1F**));**

UART0**->**BDH **=** temp **|** UART\_BDH\_SBR**(((**sbr **&** 0x1F00**)** **>>** 8**));**

UART0**->**BDL **=** **(**uint8\_t**)(**sbr **&** UART\_BDL\_SBR\_MASK**);**

UART0**->**C4 **|=** UART0\_C4\_OSR**(**osr**);**

//Mentinem majoritatea pe default

UART0**->**C1 **=** 0**;**

UART0**->**C3 **=** 0**;**

UART0**->**MA1 **=** 0**;**

UART0**->**MA2 **=** 0**;**

UART0**->**S1 **|=** 0x00**;**

UART0**->**S2 **=** 0x00**;**

//Activam UART0

UART0**->**C2 **|=** UART0\_C2\_TE\_MASK **|** UART0\_C2\_RE\_MASK**;**

**}**

void trimiteDate**(**uint8\_t data**)**

**{**

**while(!(**UART0**->**S1 **&** UART\_S1\_TDRE\_MASK**));**

UART0**->**D **=** data**;**

**}**

uint8\_t primesteDate**(**void**)**

**{**

**while(!(**UART0**->**S1 **&** UART\_S1\_RDRF\_MASK**));**

**return** UART0**->**D**;**

**}**

## **Anexa 8**

Această anexă conține codul fișierului PID.c.

#include "PID.h"

#include "Uart.h"

static float kp **=** 0**;**

static float kd **=** 0**;**

static float ki **=** 0**;**

static float eroare **=** 0**;**

static float integral **=** 0**;**

void initializarePID**(**float TS**,** float KP**,** float KI**,** float KD**)**

**{**

//Se initializeaza pidul

kp **=** KP**;**

ki **=** KI**\***TS**;**

kd **=** KD**/**TS**;**

integral **=** 0**;**

eroare **=** 0**;**

**}**

float getNextPid**(**float referinta**,** float output**)**

**{**

//Se calculeaza valoarea noua a semnalului

float up**,** ud**,** e**;**

e **=** referinta **-** output**;**

up **=** e **\*** kp**;**

integral **+=** ki **\*** **(**e **+** eroare**)** **/** 2**;**

ud **=** kd **\*** **(**e **-** eroare**);**

eroare **=** e**;**

**return** up **+** ud **+** integral**;**

**}**

## **Anexa 9**

Această anexă conține codul fișierului SenzoriDistanta.c.

#include "SenzoriDistanta.h"

#include "Uart.h"

float volatile distantaS **=** 100**;**

float volatile distantaD **=** 100**;**

float volatile distantaC **=** 100**;**

void PORTA\_IRQHandler**(**void**)**

**{**

NVIC\_ClearPendingIRQ**(**PORTA\_IRQn**);**

uint32\_t value **=** TPM0**->**CNT**;**

**if(**PORTA\_ISFR **&** 1**<<**PortSenzorDistantaCPin**)**

**{**

//Se calculeaza distanta fata de senzor

distantaC **=** FACTOR\_MUL\_DISTANTA **\*** value**;**

PORTA\_ISFR **|=** 1**<<**PortSenzorDistantaCPin**;**

distantaC **-=** DISTANTA\_EROARE**;**

**}**

#if (SENZOR\_DISTANTA\_DEBUG\_C == 1)

trimiteDate**(**distantaC**);**

#endif

**}**

void PORTD\_IRQHandler**(**void**)**

**{**

NVIC\_ClearPendingIRQ**(**PORTD\_IRQn**);**

uint32\_t value **=** TPM0**->**CNT**;**

**if(**PORTD\_ISFR **&** 1**<<**PortSenzorDistantaSPin**)**

**{**

//Se calculeaza distanta fata de senzor

distantaS **=** FACTOR\_MUL\_DISTANTA **\*** value**;**

PORTD\_ISFR **|=** 1**<<**PortSenzorDistantaSPin**;**

distantaS **-=** DISTANTA\_EROARE**;**

#if (SENZOR\_DISTANTA\_DEBUG\_S == 1)

trimiteDate**(**distantaS**);**

#endif

**}**

**if(**PORTD\_ISFR **&** 1**<<**PortSenzorDistantaDPin**)**

**{**

//Se calculeaza distanta fata de senzor

distantaD **=** FACTOR\_MUL\_DISTANTA **\*** value**;**

PORTD\_ISFR **|=** 1**<<**PortSenzorDistantaDPin**;**

distantaD **-=** DISTANTA\_EROARE**;**

#if (SENZOR\_DISTANTA\_DEBUG\_D == 1)

trimiteDate**(**distantaD**);**

#endif

**}**

**}**

void initializareSenzoriDistanta**(**void**)**

**{**

//Configurare pin trigger

PortSenzorDistantaTrig **|=** PORT\_PCR\_MUX**(**PortSenzorDistantaTrigMux**);**

PortSenzorDistantaTrig **|=** PORT\_PCR\_PE**(**1**)** **&** **(~**PORT\_PCR\_PS**(**1**));**

//Configuram pinii pentru echo

PortSenzorDistantaEchoS **|=** PORT\_PCR\_MUX**(**PortSenzorDistantaEchoSM**);**

PortSenzorDistantaEchoD **|=** PORT\_PCR\_MUX**(**PortSenzorDistantaEchoDM**);**

PortSenzorDistantaEchoC **|=** PORT\_PCR\_MUX**(**PortSenzorDistantaEchoCM**);**

PortSenzorDistantaEchoS **|=** PORT\_PCR\_PE**(**1**)** **&** **(~**PORT\_PCR\_PS**(**1**));**

PortSenzorDistantaEchoD **|=** PORT\_PCR\_PE**(**1**)** **&** **(~**PORT\_PCR\_PS**(**1**));**

PortSenzorDistantaEchoC **|=** PORT\_PCR\_PE**(**1**)** **&** **(~**PORT\_PCR\_PS**(**1**));**

PortSenzorDistantaEchoS **|=** PORT\_PCR\_IRQC**(**0xA**);**

PortSenzorDistantaEchoD **|=** PORT\_PCR\_IRQC**(**0xA**);**

PortSenzorDistantaEchoC **|=** PORT\_PCR\_IRQC**(**0xA**);**

//Dezactivare modul pe timpul configurarii

TPM0**->**SC **&=** **~(**TPM\_SC\_CMOD**(**1**));**

//Setare mod Edge-Aligned PWM

TPM0**->**CONTROLS**[**4**].**CnSC **=** TPM\_CnSC\_MSB\_MASK **|** TPM\_CnSC\_ELSB\_MASK**;**

//Setare factor de umplere

TPM0**->**CONTROLS**[**4**].**CnV **=** SENZOR\_DISTANTA\_PWM**;**

//Activare modul TPM0

TPM0**->**SC **|=** TPM\_SC\_CMOD**(**1**);**

NVIC\_SetPriority**(**PORTA\_IRQn**,** 128**);**

NVIC\_ClearPendingIRQ**(**PORTA\_IRQn**);**

NVIC\_EnableIRQ**(**PORTA\_IRQn**);**

NVIC\_SetPriority**(**PORTD\_IRQn**,** 128**);**

NVIC\_ClearPendingIRQ**(**PORTD\_IRQn**);**

NVIC\_EnableIRQ**(**PORTD\_IRQn**);**

**}**