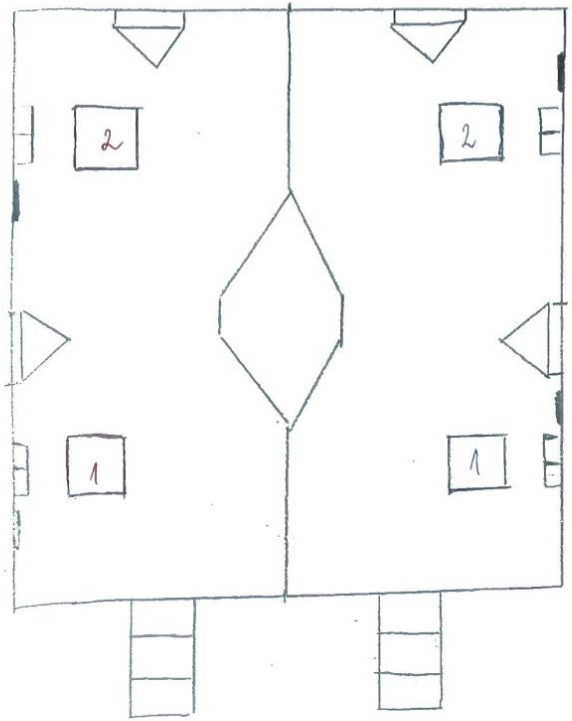
22 Februarie 2018	Explicare programe autonomie	14:00 - 20:00

Participanţi:
Costin Cristiana, Coroian Sebastian, Mathe Cezar.

Sarcini:
Explicarea tuturor programelor de autonomie

Continut:
Realizarea perioadelor de autonomie corespondente fiecărei poziţii din teren În urma ataşării encoderelor, următorul pas este programarea perioadelor de autonomie, pentru fiecare din cele 4 poziţii iniţiale posibile. Pentru fiecare dintre poziţiile iniţiale posibile, intenţionand să eliminăm mingea de culoarea alianţei opuse, vom scana pictograma VuMark, după care robotul se va îndrepta către cryptobox şi va plasa cubul în poziţia indicată pe VuMark. La final, acesta va parca în safe zone, ceea ce înseamnă un total de 85 de puncte în perioada autonomă. <u>Următoarele componente au fost utilizate în perioada autonomă:</u> 4 x Andymark Neverest 40 Gear Motor with encoders; 2 x Deluxe HiTec HS-485HB; 1 x Tetrix CR Servo 39177; 1 x MR i2c Color Sensor; 1 x MR i2c Integrating Gyro; 1x MR Range Sensor. În imaginea de pe pagina următoare e realizata o schiţă în care este explicat modul de funcţionare a fiecărei autonome, în funcţie de poziţia iniţială de pe teren. De această schiţă ne-am folosit la crearea codului sursă.



Autonomous Period

Blue 1

INIT POSITION → colorServo.setPosition(1) → detect ball color →
move → detect VuMark → move forward // run to Position →
rotate Left → move Backward → ramp1 & ramp2. set Position(1) →
ramp1 & ramp2. set Position(0) → move forward → *detect Qubes →
take Qubes → move backward → ramp1 & ramp2. setPosition(1) →
ramp1 & ramp2. set Position(0) → END

Blue 2

INIT POSITION → colorServo. set Position(1) → detect ball color →
move → detect VuMark → move forward // run to Position → (rotate 180) →
move Left → ramp1 & ramp2. set Position(1) → ramp1 & ramp2 set Pos(0) →
move Left → move backward → *detect Qubes → take Qubes →
move forward → move right → ramp1 & ramp2. set Position(1) →
ramp1 & ramp2. set Position(0) → END

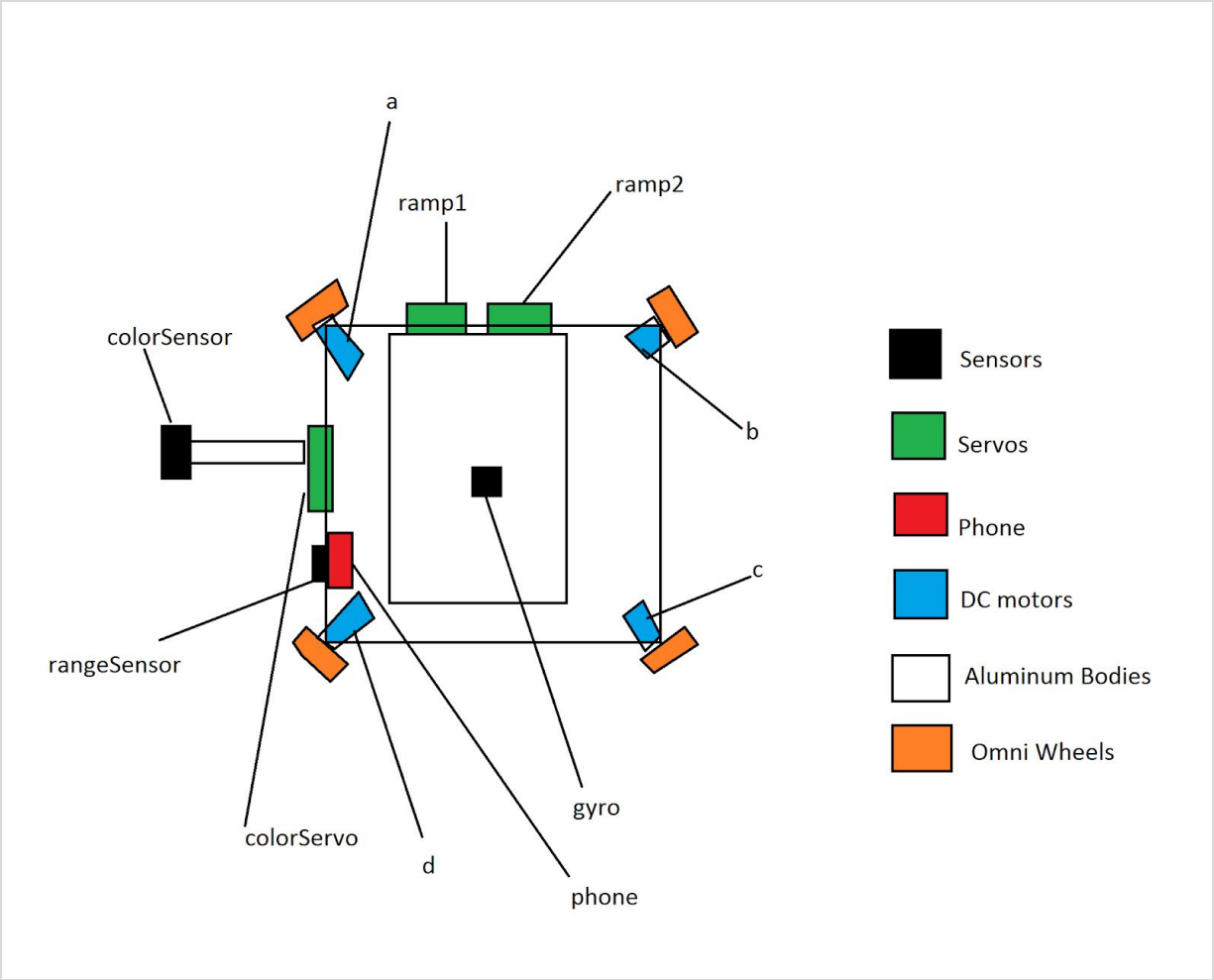
Red 1

INIT POSITION → colorServo. set Position(1) → detect ball color →
move → detect VuMark → move Backward // run to Position →
rotate Left → move Backward → ramp1 & ramp2. set Position(1) →
ramp1 & ramp2. set Position(0) → move forward → *detect Qubes →
take Qubes → move Backward → ramp1 & ramp2. set Position(1) →
ramp1 & ramp2. set Position(0) → END

Red 2

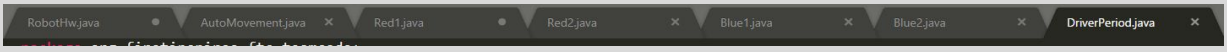
INIT POSITION → colorServo. set Position(1) → detect ball color →
move → detect VuMark → move Backward // run to Position →
move left → ramp1 & ramp2. set Position(1) → ramp1 & ramp2 set Pos(0) →
move Left → move forward → *detect Qubes → take Qubes →
move Backward → move Right → ramp1 & ramp2. set Position(1) →
ramp1 & ramp2. set Position(0) → END

Componentele mecanice ale robotului au corespondente variabilele în cod descrise în figura următoare:



Robotul dispune de un sistem de deplasare omnidirecţional de tip 4WD (4 wheels drive)
Din această cauză am decis să nu inversăm sensul niciunui motor.

În total există 7 fişiere cu extensia ".java" încărcate pe Robot controller, aşa cum se poate observa în imaginea următoare:



În continuare, voi rezuma rolul fiecărui fişier:

RobotHW.java:
Fişierul dispune de o clasă pentru iniţializarea elementelor de hardware. Aceasta este inclusă în fiecare driving mode, pentru a uşura munca programatorului.

AutoMovement.java:
Acest fişier conţine o clasă în care sunt incluse funcţiile de deplasare a robotului în perioada autonomă. Aceste funcţii se referă la mişcarea pe fiecare direcţie, rotaţie, calcularea tick-urilor motorului şi setarea vitezei de mişcare.

Red1.java:
Perioada autonomă în care robotul se află în alianţa roşie pe poziţia 1, asa cum este descris în schema de mai jos. Aceasta autonomă realizează un scor de 85 de puncte în condiţii

optime.

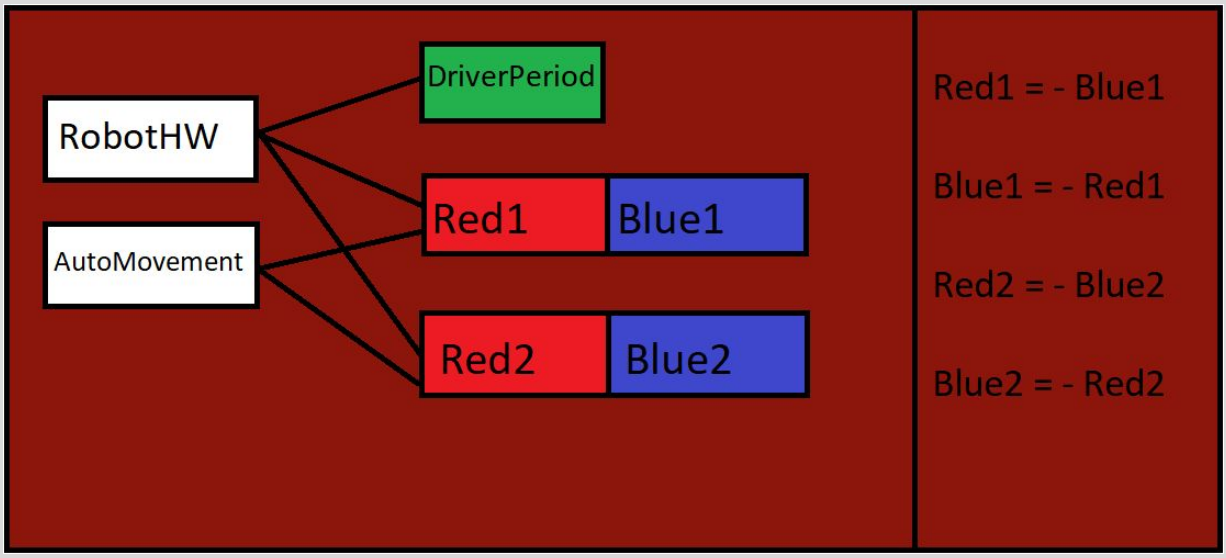
Red2.java:
Perioada autonomă în care robotul se află în alianţa roşie pe poziţia 2, asa cum este descris în schema de mai jos. Aceasta autonomă realizează un scor de 85 de puncte în condiţii optime.

Blue1.java:
Perioada autonomă în care robotul se află în alianţa albastră pe poziţia 1, aşa cum este descris în schema de mai jos. Aceasta autonomă realizează un scor de 85 de puncte în condiţii optime.

Blue2.java:
Perioada autonomă în care robotul se afla în alianţa albastra pe poziţia 2, aşa cum este descris în schema de mai jos. Această autonomă realizează un scor de 85 de puncte în condiţii optime.

DriverPeriod:
Perioada non- autonomă. Aceasta dispune de propriul driving-mode, construit pe evenimente generate în urma acţionărilor controalelor gamepad-ului. Mişcarea, precum şi funcţiile alăturate pentru completarea celorlalţi itemi ai jocului, sunt bazate pe loop-ul principal, neutilizând alte structuri de tip repetitiv.

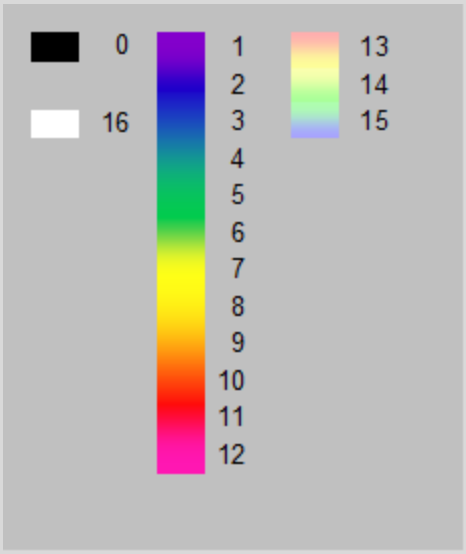
În imaginea de mai jos este reprezentată legătura între fişiere:



După cum se poate observa, DriverPeriod nu este conectat de auto movement, deoarece foloseşte valorile date de cheile gamepad-ului, nu unele predefinite, precum în autonome. De asemenea, acesta este independent de alianţă.

Totodată, autonomele Red1 şi Red2 sunt opusele autonomelor Blue1 si Blue2, ceea ce înseamnă că funcţionează pe aceleaşi principii cu extensia anumitor valori opuse.

Spre exemplu, la detectarea culorii mingii, culorile rosu şi albastru sunt complementare, aşa cum se poate vedea în imaginile de mai jos. Pentru aceste culori, senzorul de la Modern Robotics asociază valoarea intensităţii culorii reflectate, cu nişte indici de culoare, asa cum se vede în figura 2. Culoarea roşu are corespondentul 10, în timp ce culoarea albastru are corespondentul 3. Complementaritatea acestora poate fi demonstrată, pe inelul culorilor RGB (fig. 2), prin distanţa dintre indicii culorilor albastru şi roşu (3 se afla la două poziţii distanţă faţă de extremitatea 1, iar 10 se afla tot la două poziţii distanţă fata de extremitatea 12. De asemenea, distanţa faţă de punctele de centru este egală, astfel dovedindu-se complementaritatea).



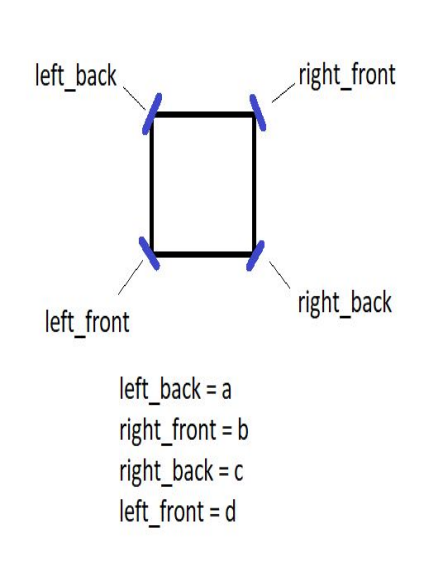
Pe lângă valorile opuse ale culorilor, autonomele blue şi red mai sunt opuse şi din punct de vedere al direcţiilor de deplasare, pentru fiecare dintre valorile de deplasare adăugându-se un “-”.
În următoarele pagini, aş vrea să prezint pe larg conţinutul fiecărui fişier.

RobotHw.java
După cum spuneam, acest fişier este iniţializarea hardware pentru toate modurile de control.

În primele linii ale clasei principale se regăsesc declaraţiile de motoare, servomotoare şi senzori utilizaţi.

```
// hardware map for configuring all hardware device
public HardwareMap hardwareMap =  null;

// drive train motors
public DcMotor left_back = null;
public DcMotor left_front = null;
public DcMotor right_front = null;
public DcMotor right_back = null;

// other motors
public DcMotor left_qubes = null;
public DcMotor right_qubes = null;
public DcMotor lift = null;
public DcMotor relic = null;

// sensors
public ModernRoboticsI2cColorSensor colorSensor = null;
public ModernRoboticsI2cGyro gyro = null;

// servos
public Servo colorServo = null;
public Servo ramp1 = null;
public Servo ramp2 = null;
```

Motoarele left_back, left_front, right_front si right_back sunt corespondentele motoarelor a, b, c şi d aşa cum este ilustrat în figura următoare:



Pe linia următoare urmează funcţia constructor, după care urmează o funcţie-argument, care efectuează efectiv iniţializarea hardware.

Funcţia este de tip public şi are ca parametru un obiect din clasa creata cu acest scop.

Pe rând, se iniţializează motoarele cu rol de drive (motoare care sunt necesare şi în perioada autonomă), după care se iniţializează şi restul motoarelor utilizate în perioada driver controlled.

Motoarele left_qubes şi right_qubes au rol de intake pentru cuburi. Acestea acţionează roţile compliante din sistemul de colectare a cuburilor.

Motorul “lift” are rolul de a eleva platforma pe care sunt poziţionate până la două cuburi, pentru a completa nivelul superior al unei coloane din cryptobox.

Pe pagina următoare este prezentata funcţia.

```java
public void init(HardwareMap hwMap) {
    // hardware map aquired from the op mode
    hardwareMap = hwMap;

    // drive train motors
    left_front = hardwareMap.get(DcMotor.class, "left_front");
    left_back = hardwareMap.get(DcMotor.class, "left_back");
    right_front = hardwareMap.get(DcMotor.class, "right_front");
    right_back = hardwareMap.get(DcMotor.class, "right_back");

    // other motors
    left_qubes = hardwareMap.get(DcMotor.class, "left_qubes");
    right_qubes = hardwareMap.get(DcMotor.class, "right_qubes");
    lift = hardwareMap.get(DcMotor.class, "lift");
    // relic = hardwareMap.get(DcMotor.class, "relic");

    // servos
    ramp1 = hardwareMap.get(Servo.class, "ramp1");
    ramp2 = hardwareMap.get(Servo.class, "ramp2");
    colorServo = hardwareMap.get(Servo.class, "colorServo");

    // sensors
    colorSensor = hardwareMap.get(ModernRoboticsI2cColorSensor.class, "colorSensor");
    gyro = hardwareMap.get(ModernRoboticsI2cGyro.class, "gyro");
}
```

Liniile următoare conţin două funcţii pentru configurarea motoarelor de mişcare, atât pentru perioada non-autonomă cât şi pentru cea autonomă.

```java
// configure the drive train to work in the driver period
public void configDriverPeriod() {
    left_front.setDirection(DcMotor.Direction.FORWARD);
    right_front.setDirection(DcMotor.Direction.FORWARD);
    left_back.setDirection(DcMotor.Direction.FORWARD);
    right_back.setDirection(DcMotor.Direction.FORWARD);
}

// configure the drive train to work in the autonomous period
public void configAutoPeriod() {
    left_front.setDirection(DcMotor.Direction.FORWARD);
    right_front.setDirection(DcMotor.Direction.FORWARD);
    left_back.setDirection(DcMotor.Direction.FORWARD);
    right_back.setDirection(DcMotor.Direction.FORWARD);
}
```

Cu toate că fiecare motor are ca direcţie implicită în configurare “FORWARD” şi funcţiile sunt identice, am decis să facem două funcţii pentru a menţine estetica şi sintaxa codului sursă, dar şi pentru a ne face configurarea mai uşoară în cazul în care programul solicită modificări sau în caz că ceva merge prost ca backup.

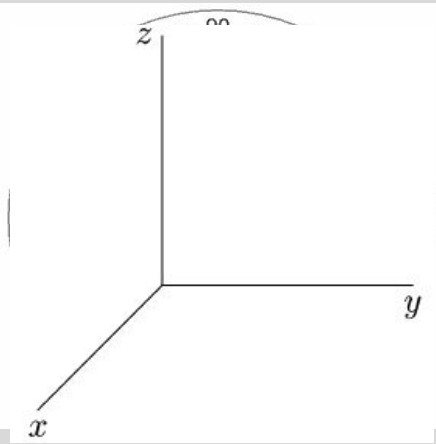
Pentru inițializarea encoderelor am realizat o funcție cu un parametru de tip boolean. În funcție de valoarea parametrului (TRUE sau FALSE), aceasta va utiliza sau nu va utiliza encodere pe motoarele de mișcare.

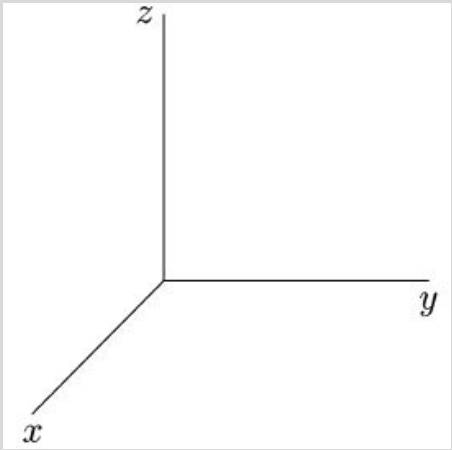
```
public void useEncoder(boolean x) {
    left_front.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    right_front.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    left_back.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    right_back.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    if (x) {
        left_front.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        right_front.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        left_back.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        right_back.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        return;
    }
    left_front.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODER);
    right_front.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODER);
    left_back.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODER);
    right_back.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODER);
    return;
}
```

Înainte de funcţia destructor, am rescris funcţia de heading pentru gyroscop in acord cu propriile noastre preferinţe și pentru a menţine liniile de cod cat mai scurte.

```
// quick function that returns the robot's rotation
public int getRotation(){
    return gyro.getHeading();
}
```

Această funcție ne permite să citim rotația robotului în funcție de axa “z” a senzorului gyro. Funcţia ia valori de la 0 la 359 întrucât, cum se poate observa în figura de mai jos, gradaţiile 0 și 360 se regăsesc în același punct.





În funcție de rotaţia faţă de poziţia inițială de pe axa Z, se poate citi variabila “heading”, ilustrând numărul de grade cu care robotul s-a întors faţă de începutul înregistrării acestora.

AutoMovement.java

Acest package conţine o clasă care include proprietăţile de deplasare a robotului în perioada autonomă.

Pe primele linii iniţializăm hardware-ul pentru rularea funcţiilor. Imediat după, memorăm într-o variabilă numărul de tick-uri necesare pentru o rotaţie completă a motorului.

```
// this is the hardware that we're controlling
RobotHw robot = new RobotHw();

// number of encoder ticks a full rotation of a wheel has
public static final int ROTATION = 1120;
```

Motoarele folosite sunt Andymark Neverrest 40 Gear Motor.
Asa cum se precizează pe site-ul oficial (www.andymark.com), pentru o rotaţie completă sunt necesare 1120 de tick-uri.

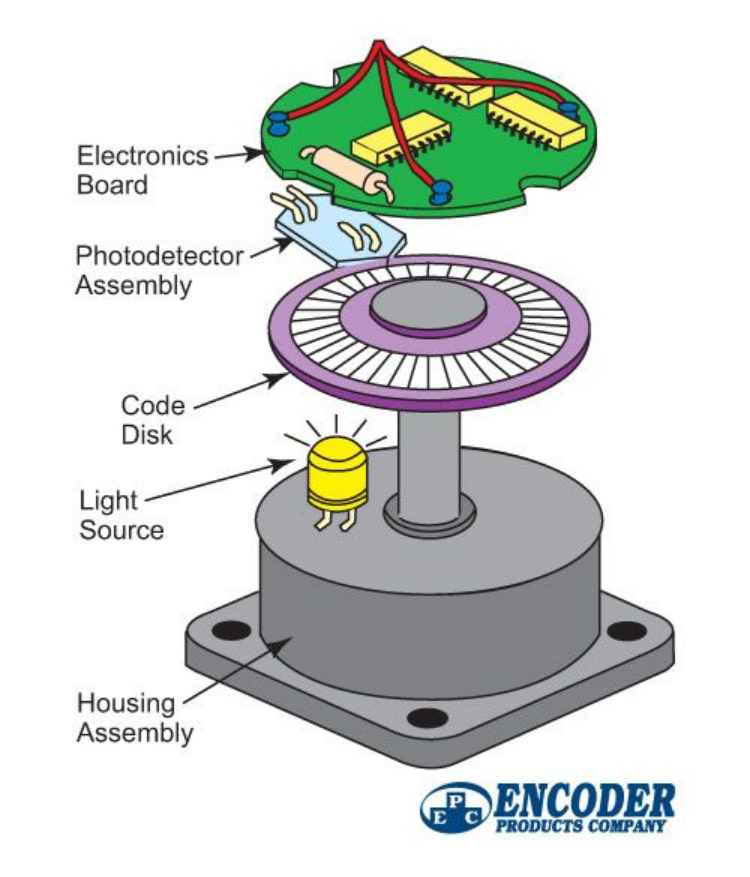
- Output counts per revolution of Output Shaft (cpr): 1120 (280 rises of Channel A)
- Output pulse per revolution of encoder shaft (ppr): 28 (7 rises of Channel A)

Pentru a memora citirea tick-urilor de pe encoder, am decis să le stocăm în variabile separate.

```
// values for the current value of each encoder of the drive train
public int left_front_encoder;
public int left_back_encoder;
public int right_front_encoder;
public int right_back_encoder;
```

Pentru aceste variabile am creat funcţii separate. Prima funcţie, cea de iniţializare, configurează poziţia iniţială în funcţie de elementele configuraţiei.
Cea de a doua funcţie are rolul de a actualiza valorile la fiecare parcurgere a funcţiei principale de loop.

```
// initialize the autonomous movement by providing the hardware
public void init(RobotHw r) {
    robot = r;
    left_front_encoder = robot.left_front.getCurrentPosition();
    left_back_encoder = robot.left_back.getCurrentPosition();
    right_front_encoder = robot.right_front.getCurrentPosition();
    right_back_encoder = robot.right_back.getCurrentPosition();
}

// function to refresh the encoder reading variables
public void refresh() {
    left_front_encoder = robot.left_front.getCurrentPosition();
    left_back_encoder = robot.left_back.getCurrentPosition();
    right_front_encoder = robot.right_front.getCurrentPosition();
    right_back_encoder = robot.right_back.getCurrentPosition();
}
```



În imaginea alaturata, preluată de pe site-ul encoder.com, este reprezentat modul de funcţionare a unui mecanism de acest fel. Aceasta ilustraţie ne este utilă pentru a înţelege mai bine codul.

Un fascicul de lumină emis de un LED trece prin Discul de cod, care este modelat cu linii opace (la fel ca spiţele pe o roată de bicicletă).

Odată ce arborele codificatorului se roteşte, fasciculul de lumină de pe LED este întrerupt de liniile opace de pe discul de cod, înainte de a fi preluat de către ansamblul fotodetector.

Acest lucru produce un semnal puls: lumină = pornit; lumină = oprit.

Semnalul este trimis către contor sau controler, care va trimite apoi semnalul pentru a produce funcţia dorită.

Datorită acestui mecanism, suntem capabili să realizăm mişcări de revoluţie de precizie ale motoarelor, care, în componenţa unor mecanisme mai mari, determina uniformizarea rotaţiilor motoarelor.

Motivul utilizării encoderelor în perioada autonomă se datorează poziţiilor fixe în care robotul trebuie să ajungă.

În cazul utilizării unei funcţii de tip “setPower(int x)” motoarele se vor învârti cu o viteza care poate lua valori de la 0 la 1. Acest interval este corespondentul numărului minim şi maxim de volţi care pot fi aplicaţi motorului. Prin urmare, în funcţie de nivelul de încărcare a acumulatorului, rotaţia poate devia. Cu cât bateria este mai descărcată, cu atat motoarele se vor roti mai încet. Acest lucru este ineficient pentru perioada de autonomie, nivelul de încărcare a bateriei fiind diferit la fiecare rulare.

Prin utilizarea encoderelor, robotul va acţiona la fel de fiecare data, indiferent de procentajul de încărcare a bateriei.

Pentru precizia funcţionării, am abordat şi alte metode, precum recunoaşterea nanometrică a obiectelor prin camera telefonului, poziţia determinată de schimbarea valorii senzorilor de distanţă, dar şi calcularea poziţiei în funcţie de acceleraţia pe axe conferita de giroscop.

În urma testelor efectuate, am decis că folosirea encoderelor este cea mai eficientă variantă pentru obiectivele pe care ni le propunem în perioada de funcționare autonomă a robotului.

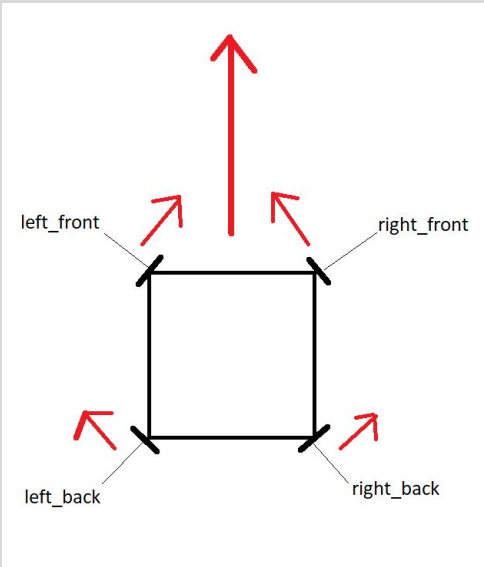
Pentru a acționa, encoderele solicită o “poziție țintă” la care să ajungă, dar și o putere cu care să se învârtă pentru a atinge poziția țintă, așa cum se poate vedea în imaginea de mai jos.

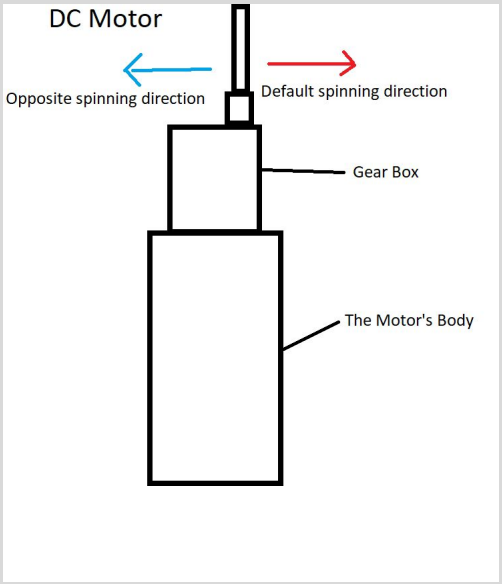
```
// function to set a target(a number of encoder ticks that a motor has to get to)
// the values are given according to a clockwise arrangement of the motors
// the first motor is the front motor on the left side
public void setTarget(int a, int b, int c, int d) {
    robot.left_front.setTargetPosition(a);
    robot.right_front.setTargetPosition(b);
    robot.right_back.setTargetPosition(c);
    robot.left_back.setTargetPosition(d);
}

// function to set a power to all the drive train motors
public void setPower(float a) {
    robot.left_front.setPower(a);
    robot.right_front.setPower(a);
    robot.right_back.setPower(a);
    robot.left_back.setPower(a);
}
```

Pentru funcțiile de mișcare față-spate-stânga-dreapta, fiecare motor trebuie să se învârtă într-un anumit sens.

Motoarele au o direcție standard de rotație, iar pentru a le inversa sensul de rotație putem să le schimbăm polaritatea, sau să le atribuim o valoare negativă în funcția de mișcare.
Sensul fiecărui motor în funcție de direcție este ilustrat în schema de mai jos.

Pentru fiecare roată în parte, este ilustrat cu rosu sensul de rotație necesar pentru ca toate acestea la un loc să deplaseze robotul în față.





Astfel, presupunând că direcția standard de deplasare a unui motor este dreapta (așa cum se vede în figura alăturata), observăm că motoarele din dreapta trebuie să se învârtă contrar direcției standard.

În tabelul de mai jos sunt ilustrate sensurile de rotaţie a motoarelor, pentru fiecare dintre direcţiile de deplasare:

DIRECTION / MOTOR	FORWARD	BACKWARD	LEFT	RIGHT
right_front	+	-	+	-
right_back	+	-	-	+
left_front	-	+	+	-
left_back	-	+	-	+

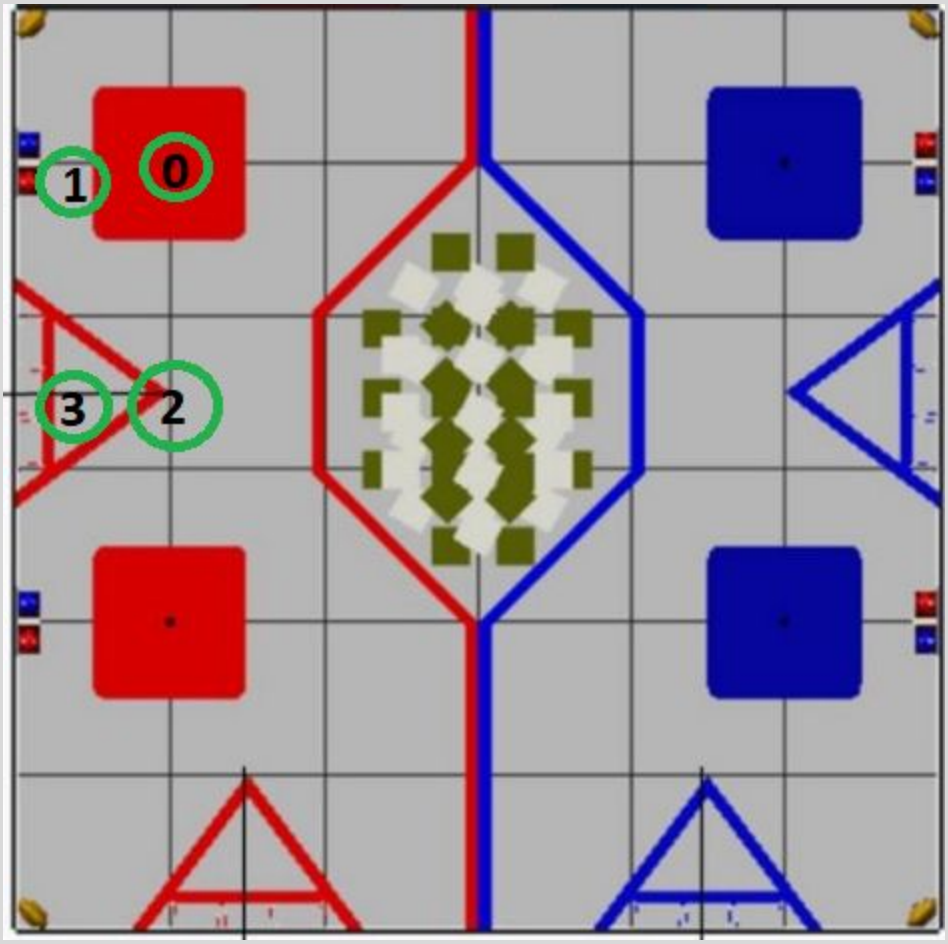
Pe baza acestui tabel am scris patru funcţii de deplasare cu aceeaşi sintaxă, dar semnele schimbate,
corespondente fiecărei linii şi coloane din tabel.

```
public void moveForward(int units) {
    refresh();
    setTarget(left_front_encoder - units, right_front_encoder + units, right_back_encoder + units, left_back_encoder - units);
    setPower(0.5f);
}

// function that moves the robot backward a number of units
// see moveForward(int) for more details aboout how this works
public void moveBackward(int units) {
    refresh();
    setTarget(left_front_encoder + units, right_front_encoder - units, right_back_encoder - units, left_back_encoder + units);
    setPower(0.5f);
}

// function that moves the robot left a number of units
// see moveForward(int) for more details aboout how this works
public void moveLeft(int units) {
    refresh();
    setTarget(left_front_encoder + units, right_front_encoder + units, right_back_encoder - units, left_back_encoder - units);
    setPower(0.5f);
}

// function that moves the robot right a number of units
// see moveForward(int) for more details aboout how this works
public void moveRight(int units) {
    refresh();
    setTarget(left_front_encoder - units, right_front_encoder - units, right_back_encoder + units, left_back_encoder + units);
    setPower(0.5f);
}
```

Pentru deplasare mai există încă o funcţie cu sintaxă diferită. Aceasta atribuie fiecărui motor o direcţie cu semnul “+”. Astfel, toate motoarele se deplasează în aceeaşi direcţie. În cazul unui parametru pozitiv, robotul se roteşte înspre dreapta la poziţia dată de valoarea parametrului, iar în cazul unui parametru negativ, se roteşte în sens opus la o poziţie egală cu modulul valorii parametrului.

```
// function that rotates the robot a number of units
// positive units rotates the robot right
// negative units rotates the robot left
// this works by giving each motor the same encoder tick it has to get to
void rotate(int units) {
    refresh();
    setTarget(left_front_encoder - units, right_front_encoder - units, right_back_encoder - units, left_back_encoder - units);
    setPower(0.5f);
}
```

Red1:



Pentru autonoma din poziţia Red 1 robotul va urma traseul:

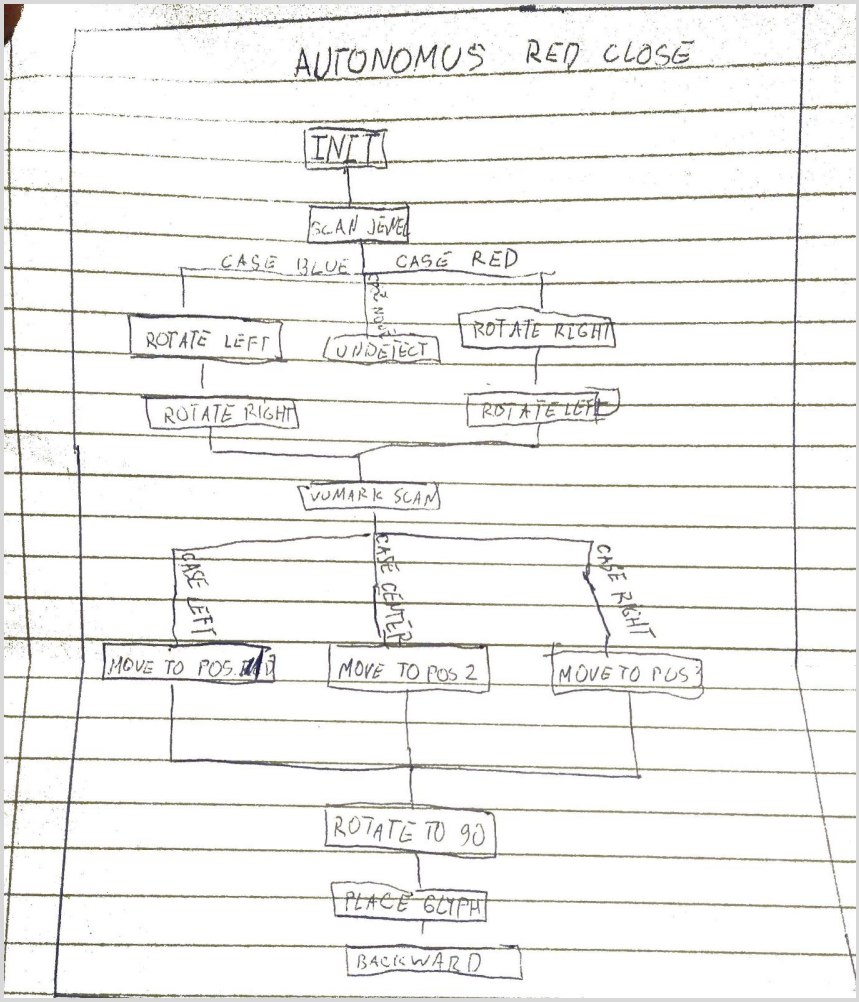
Poziţia 0 este determinată de poziţia iniţială în care robotul este plasat la începutul meciului.

Cifrele următoare sunt reprezentate de zonele în care robotul se află la diferite momente din autonomie şi efectuează o serie de acţiuni predefinite.

În zona cu numărul 1, robotul extinde profilul de care este ataşat senzorul de culoare, scanează culoarea mingii din faţa robotului şi înlătură mingea de culoarea alianţei adverse, după care scanează pictograma VuMark.

În zona 2, robotul se poziţionează în dreptul raftului din cryptobox corespondent pictogramei VuMark scanate anterior, după care execută o rotaţie de 90 de grade.

În zona cu numărul 3, robotul plasează cubul preîncărcat în cryptobox, după care se poziţionează în safe zone iar autonoma se opreşte.



Schiţa alaturata reprezintă procesul prin care algoritmul va trece pentru a îndeplini obiectivele din perioada autonomă.

Fiecare pas este marcat prin cate un bloc corespondent unui şir de sarcini.

Schiţa este realizată înainte de scrierea efectivă a codului, aşa că unii paşi nu corespund cu varianta finală.

Schiţa a reprezentat un bun suport în momentul în care am scris efectiv autonoma, insa am descoperit o mulţime de bug-uri, dar şi modalităţi mai bune de a acţiona pentru îndeplinirea sarcinilor robotului.

Pentru început, am declarat obiectele pentru clasele RobotHw şi AutoMovement:

```
RobotHw robot = new RobotHw();
AutoMovement movement = new AutoMovement();
```

```
private boolean job1 = false;
private boolean job2 = false;
private boolean job3 = false;
```

Pentru a evita utilizarea structurilor repetitive, am utilizat variabile de tip bool, cu scopul de a verifica îndeplinirea unei sarcini. De-a lungul funcţiei loop se verifică dacă sarcinile sunt completate. În cazul în care nu sunt completate, robotul execută comenzile din interiorul structurii. Dacă un job de rank inferior nu este îndeplinit, algoritmul nu va trece la complectarea unuia de rang superior. Rank-ul unei sarcini este egal cu numărul punctului de pe schiţă.

```
private boolean stage1 = false;
private int color = 0;
private boolean servoState = false;
```

Variabila stage1 reprezinta starea de completare a unui complex de sarcini, anume job1 şi job2. Dacă job1 nu este complet, algoritmul nu va trece trece la job2. Dacă job2 e complet, înseamnă că şi job1 e de asemenea complet, iar variabilei stage1 i se va atribui valoarea “true”. Dacă stage1 e complet, atunci se poate trece la job3.

Variabila color poate lua valorile 0,1 sau 3. Valoarea 0 înseamnă că nici o culoare de alianţă nu a fost detectată de către senzor. Valoarea 1 înseamnă detectarea culorii roşii, iar 2 detectarea culorii albastre.
Variabila servoState determină pozitia servo-ului de care este ataşat senzorul de culoare. Aceasta ia valoarea “true”, dacă senzorul este în modul activ şi “false”, dacă e în modul pasiv.

```
private int detectTakes = 0;
private int vuforiaTakes = 0;
```

Variabila detectTakes reprezinta numărul de scanări ale senzorului de culoare. Variabila vuforiaTakes reprezinta numărul de scanări asupra pictogramei VuMark. Fiecare dintre aceste două variabile trebuie incrementate până la 100. Astfel putem fi siguri că nu există erori în detectarea culorii corecte şi decodarea corectă a coloanei cheie, detectarea care obţine o valoare majoritară fiind cea corectă.

```
private RelicRecoveryVuMark captured = RelicRecoveryVuMark.UNKNOWN;
```

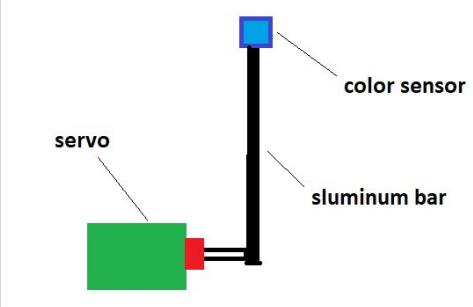
Initializam un obiect de tipul RelicRecoveryVuMark, pentru a memora detectia VuMark-ului. O initializam cu tipul de VuMark “UNKNOWN”

Functia urmatoare ne returneaza culoarea aliantei sub forma de byte, asa cum este definita in senzorul de culoare Modern Robotics

```
private int readColor() {return robot.colorSensor.readUnsignedByte(ModernRoboticsI2cColorSensor.Register
```

```
private void setServoState(boolean pos) {
    servoState = pos;
    if (servoState) {
        robot.colorServo.setPosition(0.15f);
    }
    else {
        robot.colorServo.setPosition(1f);
    }
    sleep(250);
}
```

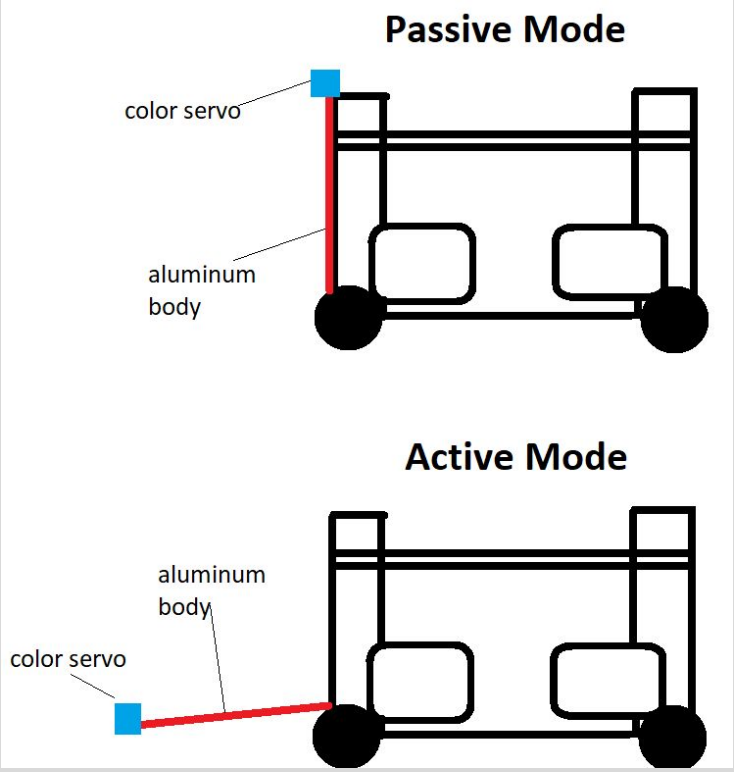
Pentru a simplifica structura codului, ne-am definit o functie dupa bunul plac, cu rolul de a seta servo-ul senzorului de culoare in mod activ sau pasiv. Valorile au fost rezultatul testelor, astfel ca senzorul sa nu aiba coliziuni cu podeaua, evitand posibilitatea de deteriorare in urma impactului



Asa cum reiese din schema, senzorul prins de un

profil de aluminiu executa o miscare de tip “sus-jos, ” atunci cand servo-ul, de asemenea atasat la profilul de aluminiu, executa functia prezentata anterior

In imaginea din dreapta se poate observa modul pasiv si modul activ in care servo-ul actioneaza.



Conform functiei, pentru valoarea “true”, pozitia servo-ului se seteaza pe valoarea flotanta 0.15, lar pentru “false” valoarea flotanta 1.

Servo-ul poate sa se roteasca pe un arc de 200 de grade. In cod, acesta se poate deplasa de la 0 la 1, 0 fiind corespondentul a 0 grade fizice iar 1 corespondentul a 200 de grade fizice, fiecare grad fiind reprezentat de o valoare egala cu 1/200 unitati

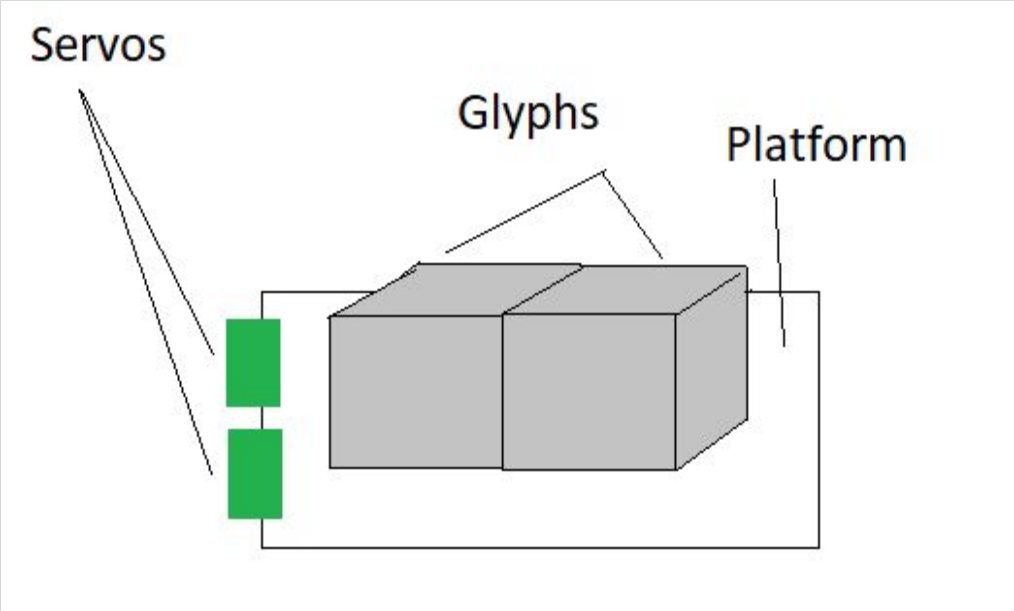
Sistemul robotului de plasat cuburi se bazeaza pe o platforma care ridica pana la doua glyph-uri. Platforma este actionata de doua servomotoare care se invart in acelasi sens, conform functiei de mai jos

```
private void qubeServos(boolean ramp){
    if (ramp == true){
        robot.ramp1.setPosition(0.05);
        robot.ramp2.setPosition(0.05);
    }
    else{
        robot.ramp1.setPosition(1);
        robot.ramp2.setPosition(1);
    }

}
```

Pentru a evita tensionarea servo-urilor atunci cand acestea se afla in modul pasiv, pozitia “true” se afla cu 10 grade mai sus de baza

robotului (0.05 unitati), astfel evitand riscul de ardere a motorului interior, dar si cel de

deteriorare a sistemului de roti zimtate din cutia de viteze.



In partea stanga este ilustrat sistemul. Servo-urile sunt atasate in aceeasi directie, deci au acelasi sens.

Servo-urile eleveaza platforma, iar cubuurile sunt plasate in cryptobox.

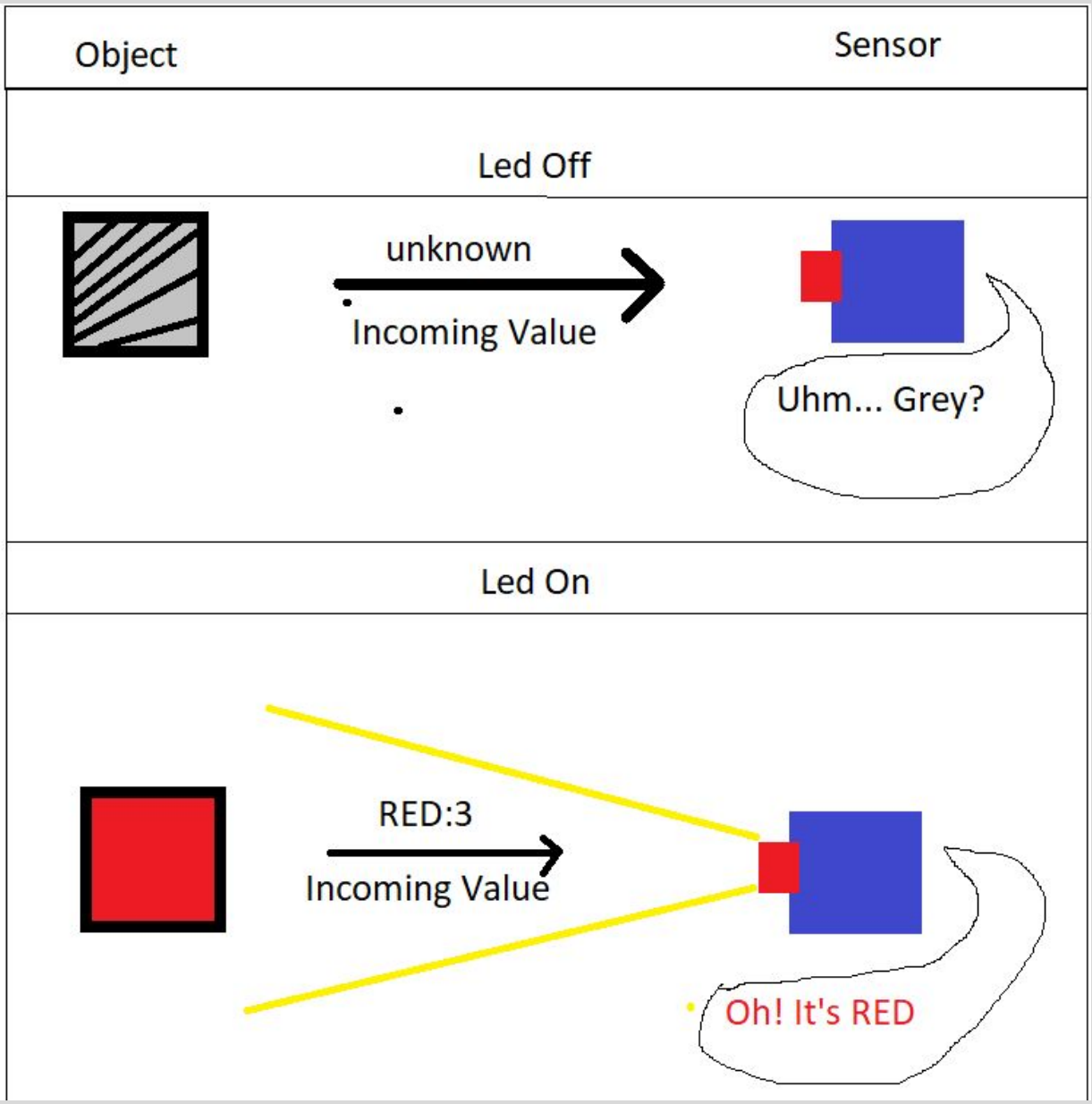
De asemenea, de platforma avem atasat si un lift pentru completarea etajului superior, dar acesta nu are rol in perioada de autonomie

```
robot.init(hardwareMap);
robot.configAutoPeriod();
robot.useEncoder(true);
movement.init(robot);

setServoState(false);
qubeServos(false);

//colorServo.setPosition(1f);
robot.colorSensor.enableLed(true);

int readcol;
```

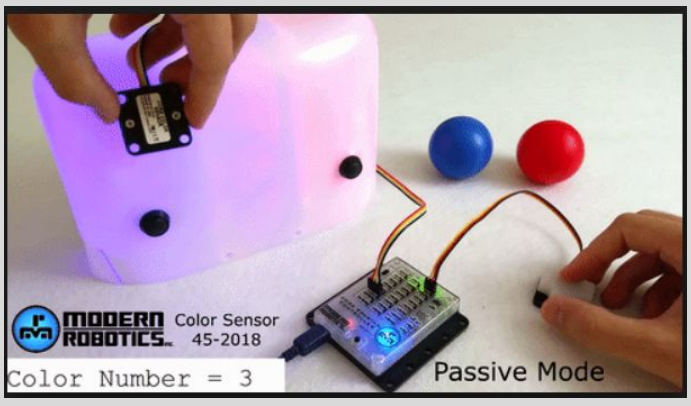
Pentu inceput, in functia de initializare, configuram hardware-ul pentru perioada autonoma si definim encoderele, asa cum sunt specificare in RobotHw.

Totodata, initializam miscarea pentru perioada autonoma, dupa care ne asiguram ca servourile nu se vor misca inainte de loopul principal.

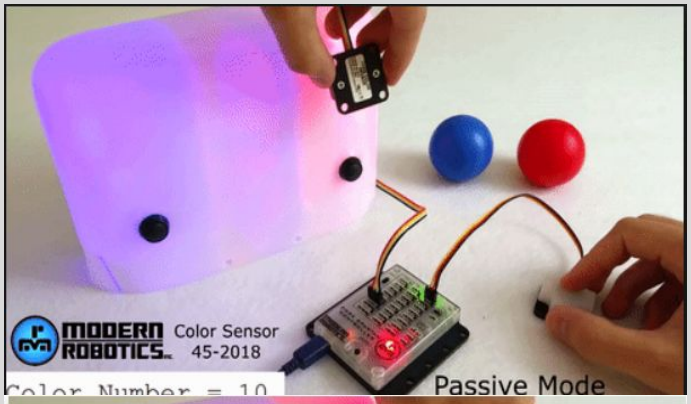
Intrucat senzorul de culoare nu primeste lumina dintr-o sursa, acesta va trebui sa citeasca lumina reflectata. Prin urmare, pentru a face obiectul sa determine lumina, acesta activeaza LED-ul, asa cum este ilustrat in figura urmatoare



In imaginile urmatoare, preluate de pe modernroboticsinc.com, se poate vedea cum senzorul returneaza diferite valori, atat cu, cat si fara led.

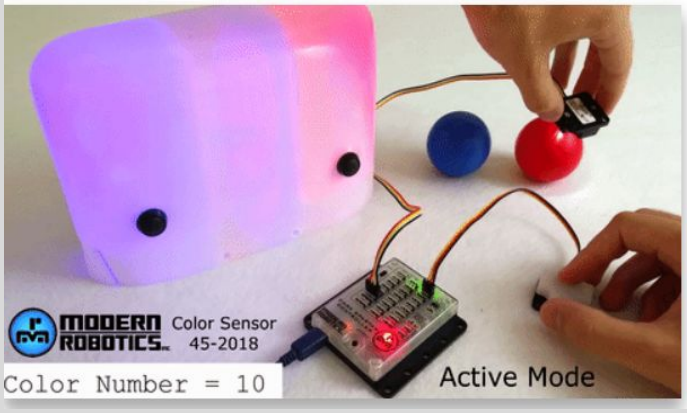


LED: OFF
Color Scanned: Blue
Type: Provided
Value Returned: 3



LED: OFF
Color Scanned: Red
Type: Provided
Value Returned: 10

LED: ON
Color Scanned: Blue
Type: Reflected
Value Returned: 3



LED: ON
Color Scanned: Red
Type: Reflected
Value Returned: 10

Pentru Vuforia, fiecare echipa FTC are o cheie unica, astfel incat aceasta sa poata fi utilizata gratuit in scop educational, ca third party software pentru aplicatia FTC Robot Controller

vuforia™

```
// VUFORIA
int cameraMonitorViewId = hardwareMap.appContext.getResources().getIdentifier("cameraMonitorViewId", "id", hardwareMap.appContext.getPackageName());
VuforiaLocalizer.Parameters parameters = new VuforiaLocalizer.Parameters(cameraMonitorViewId);
parameters.vuforiaLicenseKey = "AX9Tt1f/////AAAAmZgP0a7xc00OtCKro8AxNIB0Ga4OEzIbXne8gRzbrIWLbdDHycjj1xMWsNBKrsRDAEiWKN3LTwH77nbmYSVM4LeDtJodEejVAlG6J+
parameters.cameraDirection = VuforiaLocalizer.CameraDirection.FRONT;
this.vuforia = ClassFactory.createVuforiaLocalizer(parameters);
VuforiaTrackables relicTrackables = this.vuforia.loadTrackablesFromAsset("RelicVuMark");
VuforiaTrackable relicTemplate = relicTrackables.get(0);
```

Liniile de mai sus reprezinta initializarea acesteia si utilizarea front camera.

Odata cu executarea functiei de loop, pe ecranul de la Driver Station vor aparea o serie de date:

```
telemetry.addData("Job 1", job1);
telemetry.addData("Job 2", job2);
telemetry.addData("Job 3", job3);
telemetry.addData("Color", color);
telemetry.addData("Servo", servoState);
telemetry.addData("VuMark", captured);
```

Aceste date fac referire la completarea anumitor stadii din autonoma, dar si la culoarea detectata in cazul jewel-ului, statusul servo-ului atasat senzorului de culoare, precum si pictograma capturata de camera odata ce VuMark-ul a fost scanat.
Pentru joburi, vor aparea valorile “true” sau “false”. Pentru color, vor aparea valorile asociate culorii sau valoarea 0 in cazul nedetectarii, iar la VuMark valorile “LEFT”, “RIGHT”, “CENTER” sau “UNKNOWN” in cazul nedetectarii.

```
if (!job1) {
    if (!servoState && color == 0) {
        setServoState(true);
    }
```

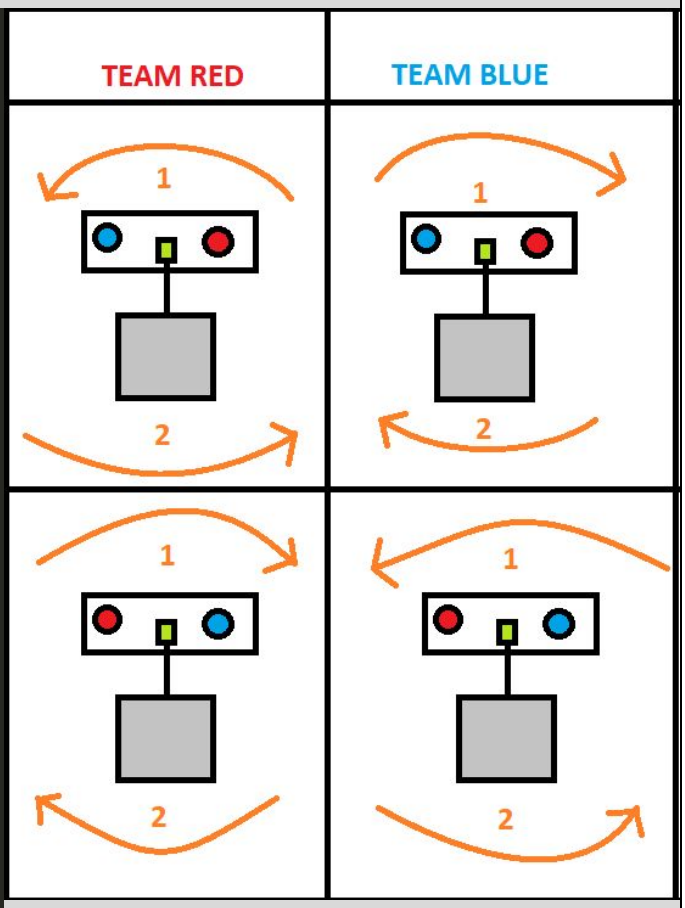
In primul rand, verificam daca primul job este complet. In cazul in care nu este, incepem prin actionarea servoului de culoare. Prima parte a acestui job este scanarea culorii, prin urmare trebuie sa scanam un jewel. Algoritmul fiind pe baza de loop, trebuie sa verificam de culoarea nu este deja scanata, acest lucru insemnand ca variabila color are valoarea initiala 0, dar si ca servo-ul nu a fost inca actionat. In urma acestei verificari, daca ambele valori din structura decizionala if corespund, vom actuiona servo-ul.

```
    else {
        if (detectTakes < 10) {
            readcol = readColor();
            if (readcol <= 11 && readcol >= 9)
                color = 1;
            else if (readcol <= 4 && readcol >= 2)
                color = 2;
            detectTakes++;
            sleep(10);
        }
```

Dupa aceasta actiune, senzorul de culoare incepe seria de scanari. Pentru a ne asigura ca nu exista mici deviatii in detectarea valorii culorii, nu am conditionat ca valoarea sa fie exacta, ci ca aceasta sa se regaseasca intr-un interval de valori, aceasta valoare fiind sensibil modificabila in functie de luminozitate si contrast.
In urma celor 10 scanari, culoarea rezultata ramane definitiva in variabila “color”.

Dupa detectarea culorii, in fuctie de valoarea acesteia, robotul efectueaza o serie de actiuni de miscare care vor indeparta jewel-ul de pe placa.
In cazul in care culoarea nu a fost detectata, robotul ignora acest sir de instructiuni.
In figura urmatoare este reprezentata directia de rotatie a robotului in functie de alianta si posibilitatile de randomizare:

```
else {
    if (color == 2) {
        movement.rotate(-300);
        movement.setPower(0.2f);
        sleep(500);
        movement.setPower(0f);
        setServoState(false);
        sleep(100);
        movement.rotate(300);
        movement.setPower(0.2f);
        sleep(500);
        movement.setPower(0f);
        sleep(100);
    }
    else if (color == 1) {
        movement.rotate(300);
        movement.setPower(0.2f);
        sleep(500);
        movement.setPower(0f);
        setServoState(false);
        sleep(100);
        movement.rotate(-300);
        movement.setPower(0.2f);
        sleep(500);
        movement.setPower(0f);
        sleep(100);
    }
```



Prin urmare, robotul executa o miscare de rotatie in directia jewel-ului de culoarea aliantei opuse, dupa care se roteste in directia opusa la pozitia initiala

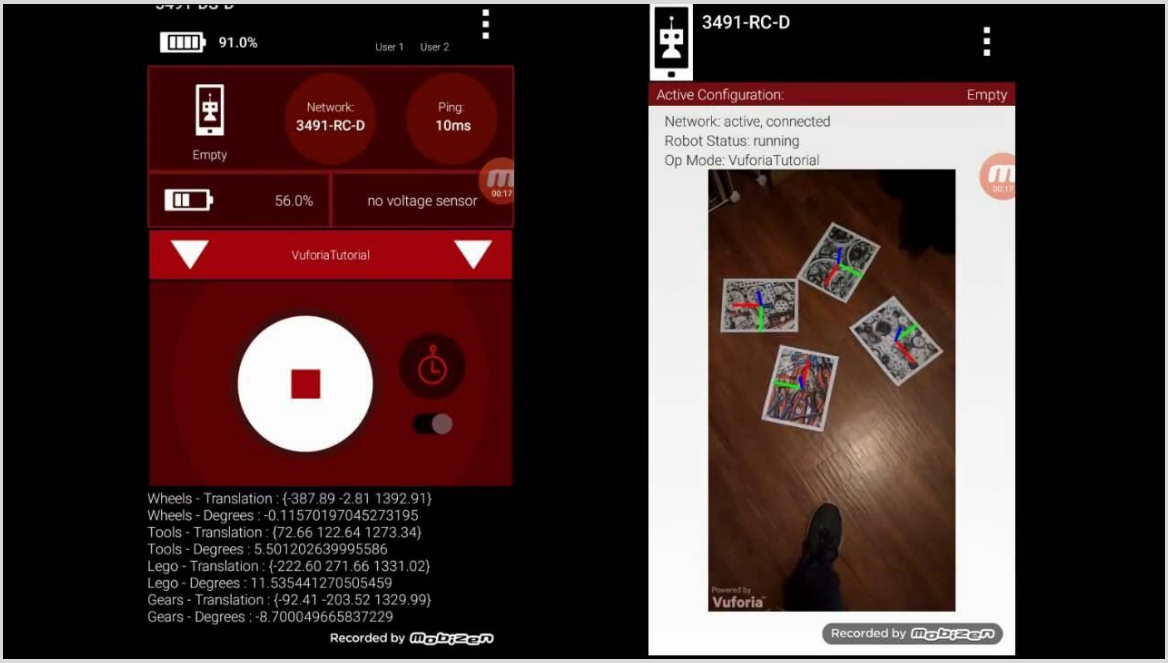
```
sleep(250);
job1 = true;
setServoState(false);
```

Dupa incheierea acestui sir de intructiuni, toate sarcinile sunt indeplinite, prin urmare variabila job1 ia valoarea true, iar servo-ul de culoare revine in pozitia pasiva, acesta neavand alt rol de-a lungul meciului.

In continuare, urmeaza job2. Acesta poate fi executat, daca nu a fost deja indeplinit in paralel cu primul job.

```
if (!job2) {
    if (vuforiaTakes < 100) {
        RelicRecoveryVuMark vuMark = RelicRecoveryVuMark.from(relicTemplate);
        if (captured == RelicRecoveryVuMark.UNKNOWN)
            captured = vuMark;
        else if (vuMark != RelicRecoveryVuMark.UNKNOWN && captured != vuMark)
            captured = vuMark;
        vuforiaTakes++;
        sleep(10);
    }
    else {
        job2 = true;
    }
}
```

In acest job, camera frontala a telefonului scaneaza, prin intermediul Vuforia, VuMarkul atasat de laterala terenului.
Odata scanat, trei axe vor aparea pe ecran, asemenea imaginii de mai jos. Imaginea provine din video-ul realizat de echipa FIXIT3491 si incarcat pe platforma YouTube, cu scopul de a



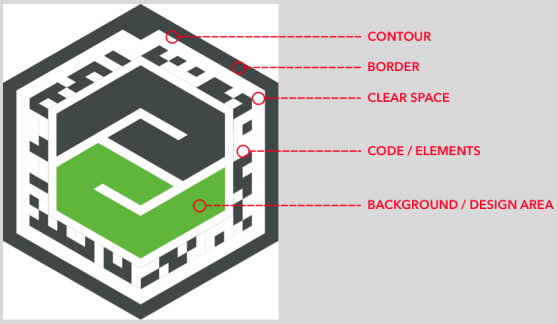
ajuta echipele incepatoare.

Pe hartiile din partea dreapta a imaginii amplasate pe podea, se pot observa cate trei axe de culori diferite (galbena, albastra si rosie).

De asemenea, in partea de stanga jos a imaginii, pot fi observate niste outputuri referitoare la denumirea modelului, pozitia si unghiul in care acestea apar pe camera telefonului.

VuMark-urile actioneaza pe acelasi principiu precum codurile QR. Acestea contin un “cifru” prescris pe o imagine complet customizabila. Cu alte cuvinte, orice imagine poate contine un VuMark.

Mai jos este prezentata structura unui astfel de VuMark, preluata de pe site-ul library.vuforia.com, precum si modele de VuMark sib forma key-urilor din jocul Relic Recovery



VuMark cu incriptia “LEFT”

VuMark cu incriptia “CENTER”

VuMark incriptia “RIGHT”

Dupa completarea scanarii VuMark-ului, valoarea acestuia se salveaza in variabila “captured”, iar programul trece mai departe

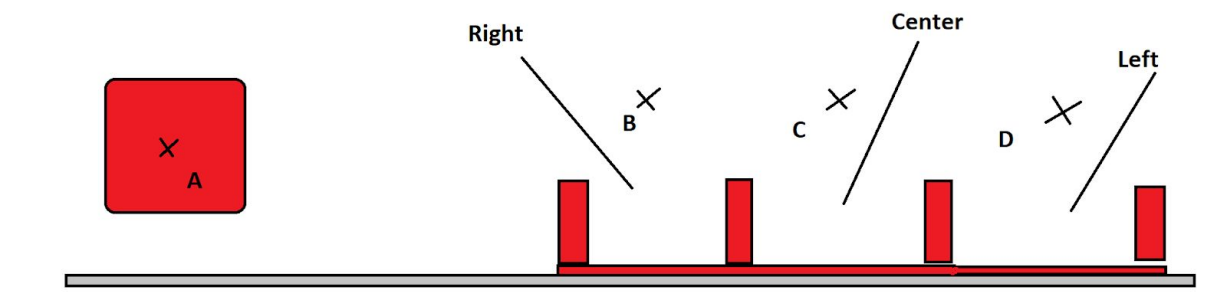
```
if (job1 && job2) {
    relicTrackables.deactivate();
    stage1 = true;
```

In cazul in care si job1 si job2 sunt completate, stadiul 1 in care robotul se afla pe balancing stone este complet. Prin urmare, variabila “stage1” primeste valoarea “true”.
Aceasta variabila este necesara pentru completarea jobului 3. Jobul 3 consta in deplasarea robotului in punctul optim corespondent cu codul descifrat din VuMark si plasarea cubului in coloana corecta, dupa care motoarele isi inceteaza activitatea in safe zone.

```
if (!job3 && stage1) {
    int distance;
    if (captured == RelicRecoveryVuMark.LEFT) {
        distance = movement.ROTATION * 272 / 100;
    }
    else if (captured == RelicRecoveryVuMark.CENTER) {
        distance = movement.ROTATION * 221 / 100;
    }
    else if (captured == RelicRecoveryVuMark.RIGHT) {
        distance = movement.ROTATION * 181 / 100;
    }
    else {
        distance = movement.ROTATION;
    }
```

Pentru inceputul jobului 3, declaram o variabila care va memora distanta (in tick-uri de motor) necesara pentru a ajunge intr-o anumita zona corespondenta coloanei din cryptobox indicate pe VuMark, dupa care se va deplasa in fata cu acea valoare

```
movement.moveForward(distance);
```



A - Starting Position - No Movement required
B- Right Cryptobox Position - 181/100
C- Center Cryptobox Position - 221/100
D- Left Cryptobox Position - 272/100

Pozitiile pot fi mai bine vizualizate in schema alaturata, care reprezinta terenul de joc vazut de sus:

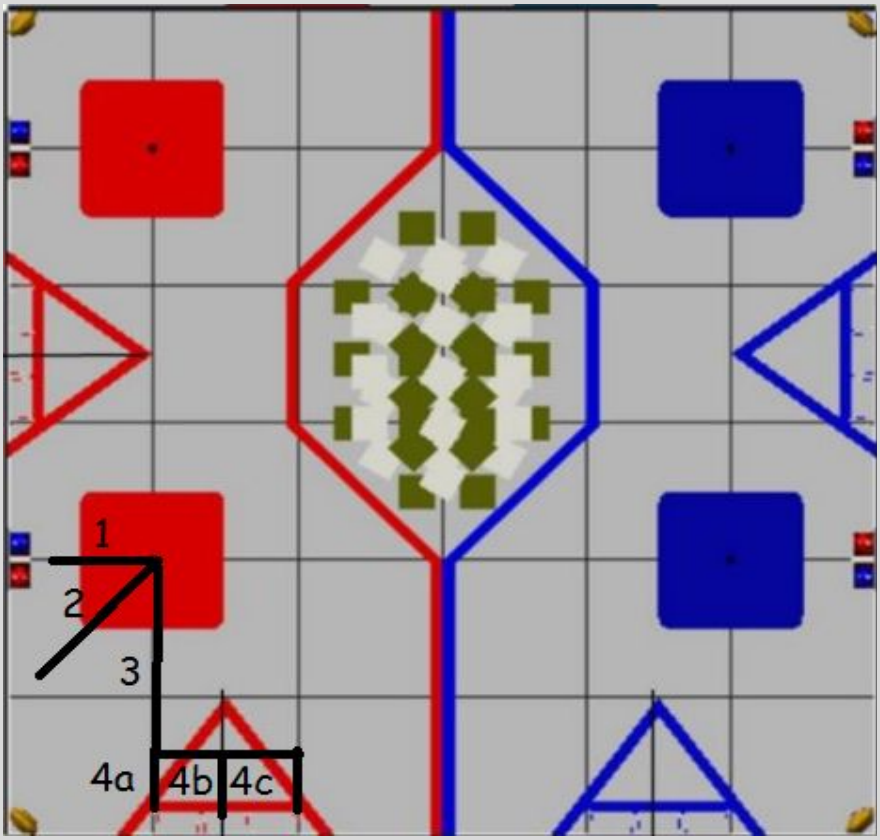
Robotul executa o rotatie de 30 de grade, astfel incat partea laterala care include platforma se afla pe acelasi plan cu coloana din cryptobox. Dupa ce robotul s-a rotit, acesta se misca incet in spate si se opreste la distanta optima pentru eliberarea glyph-ului

```
movement.rotate(-movement.ROTATION * 12 / 10);
sleep(2000);
movement.setPower(0f);
movement.moveBackward(movement.ROTATION / 2);
sleep(2000);
movement.setPower(0f);
```

```
qubeServos(true);
sleep(1000);
qubeServos(false);
job3 = true;
```

In aceasta pozitie, platforma eleveaza glyph-ul preincarcat in cryptobox, dupa care revine la pozitia initiala, iar autonoma se opreste.

Red2:
Plecand din pozitia Red2 , vom reperezenta actiunile efectuate de robot in aceasta autonoma.



1- senzorul de culoare atasat bratului va detecta culoarea Jewel-ului, iar mai apoi il va inlatura pe cel care are culoarea opusa aliantei;

2- cu ajutorul camerei frontale a telefonului, vom identifica VuMark-ul afisat;

3- robotul se va misca in fata;

4- in functie de VuMark-ul detectat, robotul se va misca la stanga si se va pozitiona in fata unui anumit cryptobox.

a- pentru VuMark-ul Right
b- pentru VuMark-ul Center
c- pentru VuMark-ul Left

Rampa se va ridica, iar glyph-ul aflat pe aceasta va fi asezat in locul potrivit.

```
RobotHw robot = new RobotHw();
AutoMovement movement = new AutoMovement();
```

Am creat obiectele pentru clasele RobotHw si AutoMovement.

Urmatoarele variabile ne vor ajuta pe tot parcusul autonomei. Am impartit codul in trei mari parti.

```
private boolean job1 = false;
private boolean job2 = false;
private boolean job3 = false;
```

← variabila pentru primul task
← variabila pentru al doilea task
← variabila pentru al treilea task

Acestea au fost initializate cu FALSE, deoarece nicio actiune nu este indeplinita initial. De asemenea, task-urile depind unele de celelalte (daca job1 inca are valoarea FALSE, job 2 nu poate fi indeplinit si, implicit, nici job3).

```
private boolean stage1 = false;
private int color = 0;
private boolean servoState = false;
```