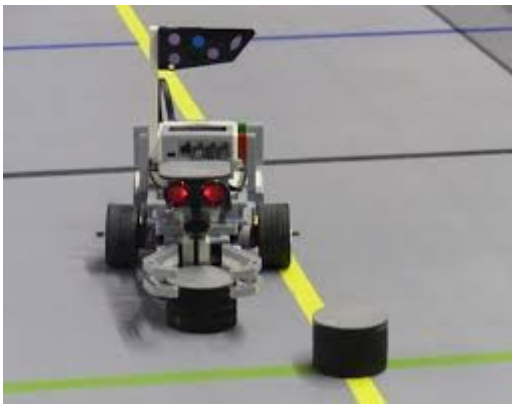
Projet de robotique
-
Lego Mindstorm



Plan de test

Membres :
-Balatti Enzo
-Bellatif Mamoune
-Bergman Clément

L3 MIASHS
Année 2019-2020

Table des matières

Introduction

Contexte

Aucune classe test n'a été crée, nous avons uniquement modifié les méthodes dans le programme principal pour tester les différents cas. Les méthodes et codes affichés ci-dessous sont les mêmes que dans le programme principal avec des affichages sur la console ainsi que des appuis sur bouton nécessaires pour continuer le programme. De plus certains test n'ont pu être effectués par manque de temps, leurs déroulement sont quand même décrits dans ce rapport.

Tests des différents capteurs

Capteur de couleur

Objectif : arriver à détecter une couleur parmis blanc,noir, jaune, bleu rouge ou vert

1-enregistrer une donnée du capteur couleur

c'est la première étape de la méthode *calibrage()*

```
float[] blue = new float[sampleCol.sampleSize()]; // creation du tableau des composantes RVB du capteur
sampleCol.fetchSample(blue, 0); // les valeurs detectées sont rangées dans le tableau col, à partir de l'indice 0
String bl= (String)""+blue[0]+"/"+blue[1]+"/"+blue[2];
```

2-écrire les données RVB dans un fichier texte spécialisé

```
try { fichierCouleurs = new
FileWriter("/home/lejos/programs/couleurs.txt");
//on crée un nouveau fichier d'ecriture.
}catch(Exception e) {System.out.println(e);}
try { fichierCouleurs.write(bl); //on écrit dans le fichier les différentes valeurs pour la couleur bleue
fichierCouleurs.write("\n");//on saute une ligne
}catch(Exception e) {System.out.println(e);}
```

3-arriver à lire le fichier de données

Ouvrir le fichier couleur.txt et voir si la chaîne bl a été écrite

4-extraire les données ligne par ligne du fichier et les stocker dans les tableaux de couleur respectifs (préalable au calcul du scalaire)

```
String[] split=new String[sampleCol.sampleSize()];
String s= bufCol.readLine(); //on extrait la ligne lue dans la chaîne s
System.out.println(s);
split=s.split("/"); //avec la méthode split, on decoupe la chaîne dans le tableau, le séparateur étant "/"
blue[0]=new Float(split[0]); // le tableau de valeurs RVB de chaque couleur est rempli par celles de split
blue[1]=new Float(split[1]);
blue[2]=new Float(split[2]);
```

5-détecter la couleur grâce à la méthode scalaire

appeler la méthode *getCouleur()* devant une ligne et vérifier si la chaîne retournée est la coueur de la ligne testée. (*TRUE* affiché dans la console

```
System.out.println(getCouleur()=="white");
```

En cas de test non concluant, on affiche tour à tour via la console le calcul du scalaire entre la couleur détectée et chaque couleur enregistrée. (Ici la démarche pour white, faire de même pour chaque autre couleur) :

```
double scalaire = scalaire(sample, white);
System.out.println(scalaire);
Button.waitForAnyPress();
if (scalaire < minscal) {
      minscal = scalaire;
      color = "white";
}
```

Capteur de pression

Objectif : savoir si le palet est entre les pinces.

On teste ici la méthode paletDansPince()

```
public boolean paletDansPince() {
samplePress.fetchSample(tabPress, 0);
return(tabPress[0]==1);//1 ==> pression activée
}
```

Capteur de distance via ultrasons

Objectif : détecter les objets à distance le plus précisément possible

1- Stocker une distance et l'afficher

On place un objet à 50cm du capteur, on affiche la distance détectée par le capteur et on compare aux 50 cm réels. La distance affichée est en mètres

```
sampleUS.fetchSample(usSample, 0); // attribution valeur capteur US à
case 0 du tableau usSample
double distance = usSample[0];
System.out.println("distance = "+distance);
```

2-Vérifier les capacités minimum et maximales de mesures totales

Même code à tester sauf que l'on le réitère jusqu'à ce que le robot retourne *infinity* pour la distance max et *0* pour la distance minimale.

3-Vérifier les capacités minimum et maximales de détection de palet

Comme pour le point précédent, mais ici l'objet est de la taille d'un palet, étant donné qu'un palet est moins haut qu'un mur ou qu'un robot, sa détection est impossible en dessous d'une certaine distance avec le capteur.

4-Stocker plusieurs valeurs dans un tableau et sélectionner la plus petite par exemple

Il s'agit là de faire un tour entier du robot, et de stocker toutes les distances. On place un objet plus près qu'un autre et on vérifie si la distance minimale détectée est bien celle séparant le capteur de l'objet le plus proche

```
ArrayList<Double> tabDisCrt =new ArrayList<Double>();
```

```
virage(360);
while(chassis.isMoving()) {
      sampleUS.fetchSample(usSample, 0); // attribution valeur capteur US
à case 0 du tableau usSample
      double distance = usSample[0];
      tabDisCrt.add(distance*1000);// on stocke les distances dans une
liste
      Delay.msDelay(1);//on passe dans la boucle toutes les millisecondes
            }
System.out.println("distance minimale =
"+tabDisCrt.get(getMin(tabDisCrt)));
```

Test des différents moteurs

Chassis

Objectif : effectuer des translations et des rotations précises.

1-avancer d'une distance et s'arrêter ensuite

On teste notre méthode *avancerDe(double distance)* (distance en millimètres)

```
avancerDe(500);
```

On vérifie si le robot a avancé de 50cm précisément.

2-reculer et s'arrêter

```
avancerDe(-500);
```

3-tourner d'un angle donné

On teste notre méthode *virage(double rotation)* (angle en degrés)

```
public void virage(double rotation){
chassis.rotate(rotation);
      }
```

4-changer de vitesse entre deux déplacements

On compare le temps que le robot met pour effectuer ces deux distances

```
virage(360);
avancerDe(500);
while(chassis.isMoving()) {}
chassis.setSpeed(500, 495);
avancerDe(500);
while(chassis.isMoving()) {}
```

Pinces

Objectif : ouvrir les pinces pour attraper un palet et les refermer ensuite

1-Connaître la valeur maximale d'ouverture des pinces et celle de fermeture.

A l'aide de l'écran de commande Start EV3 Control, contrôler manuellement l'ouverture des pinces (forward). S'arreter a l'ouverture maximale voulue, et récuperer la donnée affiché dans l'écran de contrôle de la position (tachoCount). Effectuer l'inverse pour

refermer les pinces (backward).

2-Ouvrir les pinces jusqu'à la valeur limite

```
mP.forward();
while(mP.getPosition()<700)
```

3-Fermer les pinces jusqu'à la valeur limite.

```
mP.backward();
while(mP.getPosition()>20)
```

4-test final, ramasser un palet

Il s'agit de la première version de la méthode *ramasserPalet()*

```
mP.forward(); //ouverture des pinces
while(!paletDansPince()) {
	System.out.println("open");
	if(mP.getPosition()>700) {// tant que le palet n'est pas dans les pinces et que l'ouvertur des pinces est inferieure a 700, on ouvre les pinces
	break;
	}
}
mP.stop();// on stoppe le moteur des pinces une fois la position des pinces a 700
chassis.travel(350);
while(paletDansPince()) {}
chassis.stop();// ici le robot a attrapé un palet, il s'arrete alors et ferme ses pinces
mP.backward();
while(mP.getPosition()>20) {
	System.out.println("close");
} mP.stop();
```

Test des méthodes de jeu

Méthode marquerPremierPalet

Objectif : marquer le premier but le plus rapidement possibles

1-attraper le premier palet

On teste si le robot avance jusqu'au premier palet et l'attrape

```
public void marquerPremierPalet() throws IOException {
	mP.forward(); //ouverture des pinces
	chassis.travel(600);
	while(!paletDansPince() ) {
if(mP.getPosition()>500) {// on ouvre les pinces jusqu'a ce qu'elles soient a la position 500, et tant que le palet n'est pas dans les pinces
```

```
		break;
		}
}
mP.stop();//on arrete les moteurs des pinces une fois celles ci ouvertes
		while(!paletDansPince()) {}//on continue d'avancer tant que le
palet n'est pas dans les pinces
		chassis.stop();//le robot arrete d'avancer une fois le palet saisi
		mP.backward();
		while(mP.getPosition()>10) {}// on ferme les pinces jusqu'a ce
qu'elles atteignent le position 10
		mP.stop();
```

2-Se décaler assez de la ligne de couleur pour ne pas toucher d'autre palets et être aligné

```
virage(25); // on tourne d'un angle défini, on avance de 40 cm puis on se
remet en face de l'en-but adverse pour eviter de rentrer en collision
avec d'autres palets
while(chassis.isMoving()) {}
chassis.travel(400);
while(chassis.isMoving()) {}
virage(-25);
while(chassis.isMoving()) {}
```

3-avancer jusqu'à la ligne d'en-but et marquer le but.

```
chassis.travel(2000);
while(chassis.isMoving()) {// on avance jusqu'a la ligne d'en-but adverse
		if(getCouleur()=="white") {
			chassis.stop();
		}
}
try{
		deposerPalet();
}catch(Exception e) {System.out.println(e);}
```

Rechercher le palet le plus proche

Objectif : identifier l'objet le plus proche, se positionner dans sa direction, l'attraper si c'est un palet et marquer le but.

1-identifier l'objet le plus proche

Ce premier test est similaire au test 4 du capteur de distance par ultrasons, il faut juste qu'en plus d'identifier la distance minimale, connaître son angle d'inclinaison par rapport à la position initiale du robot. On pose un palet à 30cm de distance du capteur et à sa gauche (à 90°). En face du robot, on place un autre palet mais cette fois-ci face au capteur (0°) et à 40cm du robot. Pour valider le test, le robot doit effectuer son tour et afficher dans la console une distance de 40cm et un angle de 90° (+- 3° de marge d'erreur)

```
ArrayList<Double> tabDisCrt =new ArrayList<Double>();
virage(360);
while(chassis.isMoving()) {
	sampleUS.fetchSample(usSample, 0); // attribution valeur capteur US
à case 0 du tableau usSample
	double distance = usSample[0];
	tabDisCrt.add(distance*1000);// on stocke les distances dans une
liste
	Delay.msDelay(1);//on passe dans la boucle toutes les millisecondes
		}
int index = getMin(tabDisCrt);//on recupere la valeur de l'index
correspondant a la distance de l'objet le plus proche pouvant etre un
palet
double angleMin = (double)(360.0/
(double)tabDisCrt.size())*(double)index;//on determine l'angle
correspondant a l'objet identifie comme le plus proche
double distanceMin = tabDisCrt.get(index);// on recupere la distance
existante entre le robot et l'objet le plus proche
System.out.println("distance minimale = "+distanceMin);
System.out.println("angle = "+angleMin);
Button.waitForAnyPress();
```

2-se positionner dans la bonne direction le plus rapidement possible

Il s'agit là ensuite de vérifier que le robot se décale bien de l'angle associé à la distance minimale et de la manière la plus rapide possible. C'est à dire de tourner vers la gauche quand le palet est sur la gauche et inversement.

```
if (angleMin>180)
	angleMin=-(angleMin-180);
virage(angleMin);//on s'oriente vers l'objet identifie
while(chassis.isMoving()) {};
```

3-On vérifie si le robot détermine ou non que l'objet est un palet

```
chassis.travel(distanceMin-150);//on avance jusqu'a 15 cm avant l'objet
while(chassis.isMoving()) {};
sampleUS.fetchSample(usSample, 0);
double distance = usSample[0];
if (distance*1000>170) {//si la distance par rapport a l'objet
initialement identifiée est superieure a 17 cm, alors l'objet est un
palet ou un robot, une erreur de 2 cm est anticipée sur les mesures du
robot
System.out.print("Il y a un palet");
Button.waitForAnyPress();
ramasserPalet();
marquerPalet();
}else {// si la distance est superieure a 17 cm l'objet n'est soit plus
présent, soit ce n'est pas un palet
System.out.print("Il n'y a pas de palet");
Button.waitForAnyPress();
}
```

4-le robot se replace si il va hors de la zone de recherches

La zone de recherche est est la zone entre les en-but et à l'intérieur des lignes jaunes et rouges. Pour éviter de taper dans les murs et d'être bloqué, on veut s'assurer que le robot se replace dans la zone de recherche et recommence une procédure de recherche dans ces cas.

On modifie donc le test précédent :

```
chassis.travel(distanceMin-150);//on avance jusqu'a 15 cm avant l'objet
      while(chassis.isMoving()) {
            if(getCouleur()=="white") {// si une ligne blanche est
detectee on fait demi tour et on cherche un autre palet
                  chassis.stop();
                  System.out.println("ligne blanche, demi tour");
                  Button.waitForAnyPress();
                  if((angleCrt<90 || angleCrt> 270)) { // ici la ligne
blanche detectée est celle de notre enbut
                        virage(180-angleCrt);// virage dans le bon sens
suivant les angles d'incidence
                        while(chassis.isMoving()) {}
                        chassis.travel(200);
                        System.out.println("angle courant= "+angleCrt);
                        Button.waitForAnyPress();
                  }else {// ici la ligne blanche detectée est celle de
notre ligne de départ
                        virage(-(angleCrt-180));// virage dans le bon sens
suivant les angles d'incidence
                        while(chassis.isMoving()) {}
                        chassis.travel(200);
                        System.out.println("angle courant= "+angleCrt);
                        Button.waitForAnyPress();
```

```
                }
            } else if(getCouleur()=="yellow" || getCouleur()=="red") {
//si on passe une ligne rouge ou jaune on recule et on cherche un autre
palet
                chassis.travel(-400);
                while(chassis.isMoving()) {}
                System.out.println("direction le mur, on recule");
                Button.waitForAnyPress();
            }
      }
```

Marquer palet

Certains tests de cette méthode ont déjà été fait en amont lors des tests des capteurs et moteurs. On veut vérifier si le robot atteint la ligne dans les cas où il arrive de côté et risque de ne pas atteindre l'en-but mais les mur latéraux (soit dès que le robot dépasse les lignes rouge ou jaune avec un palet dans les pinces) :

Pour ce faire on place le robot en direction d'une ligne rouge (jaune ensuite) et on lance la méthode *marquerPalet()*. Si à la détection de la ligne il pivote de 90° et direction de l'en-but, alors le test est réussi

```
chassis.travel(3000);//avancer de trois metres permet que le robot
finisse par s'arreter meme si il ne detecte pas la ligne blanche
int a=0;
while(chassis.isMoving()) {//on avance jusqu'a la ligne adverse
      if(getCouleur()=="red" ) { //si on croise une ligne jaune ou rouge
on s'arrete et on s'oriente vers la ligne d'en-but
            if(enBut=="mur") {
                chassis.stop();
                virage(-90);//le sens du virage dépend de la position de
l'enbut (côté mur ou côté salle)
                while(chassis.isMoving()) {}
            }else {
            chassis.stop();
            virage(90);
            while(chassis.isMoving()) {}
            }
a=1;// si on croise une ligne rouge ou jaune la variable a prends la
valeur 1 et on sort du while car le chassis s'est arrete
      } else if(getCouleur()=="yellow") {
            if(enBut=="mur") {
                chassis.stop();
                virage(-90);//le sens du virage dépend de la position de
```

```
l'enbut (côté mur ou côté salle)
                while(chassis.isMoving()) {}
            }else {
            chassis.stop();
            virage(-90);
            while(chassis.isMoving()) {}
            }
a=1;
      }else if(getCouleur()=="white") {
            chassis.stop();
      }
}
if (a==1) {// si a=1, on avance vers la ligne d'en-but adverse
      chassis.travel(3000);
      while(chassis.isMoving()) {
            if(getCouleur()=="white") {
                  chassis.stop();
            }
      }
}
```

Tests n'ayant pu être effectués

Procédure de replacement

Ajouter une procédure de replacement à notre programme lors du match, impliquait un nombre élevés de test que nous n'avons pu faire. Les test et codes ci-dessous on été imaginés mais jamais mis en pratique.

```
public void procedureDeReplacement()throws IOException {
faceAuMur();
virage(180);
replacement();
System.out.println("fin, angleCrt= "+angleCrt);
Button.waitForAnyPress();
chercherObstaclePlusProche();
}
```

Lorsque la procédure de remplacement est lancé il faut vérifier :

1-Tester la méthode faceAuMur()

Pour être sur que le robot s'est placé face au mur, il faut effectuer un balayage via le capteur de distance ultrason sur 180° pour ensuite tourner de l'angle où la distance en direction du mur détectée est la plus petite.

```
public void faceAuMur() {
	chassis.setAngularSpeed(60);
	ArrayList<Double> liste =new ArrayList<Double>();
	virage(180);
	while(chassis.isMoving()) {
		sampleUS.fetchSample(usSample, 0); // attribution valeur
capteur US à case 0 du tableau usSample
		double distance = usSample[0];
		liste.add(distance*1000);
		Delay.msDelay(1);
	}
	int index = getMin(liste);
	double angleMin = (double)(360.0/
(double)liste.size())*(double)index;
	double distanceMin = liste.get(index);
	System.out.println(" index ="+index +" angle="+angleMin+" distance=
"+distanceMin);
	Button.waitForAnyPress();
	virage(-180+angleMin);
	while(chassis.isMoving()) {}
	liste.clear();
	}
```

2-Tester la méthode replacement()

Cette méthode est censé remettre le robot fans la direction de son en-but, en fonction des deux lignes de couleurs qu'il a traversé et remettre la variable *angleCrt* à zéro. Il faut donc tester les différents cas où le robot serait déjà dans le même sens, l'inverse ou dans un autre sens que celui de son en-but, les voici :

Dans le cas où l'en but est côté *salle*, la méthode *replacement()* doit permettre,

-De ne rien faire si les couleurs des lignes détectées sont verte puis noire, noire puis bleue, blanche puis verte ou bleue puis blanche.

-De faire demi-tour si les couleurs des lignes détectées sont noire puis verte, bleue puis noire, verte puis blanche ou blanche puis bleue.

-De pivoter de 90° vers la droite si les couleurs détectées sont jaune puis noire ou noire puis rouge.

-De pivoter de 90° vers la gauche si les couleurs détectées sont rouge puis noire ou noire puis jaune.

Dans le cas où l'en but est côté mur, la méthode replacement() doit permettre,

-De ne rien faire si les couleurs des lignes détectées sont noire puis verte, bleue puis noire, verte puis blanche ou blanche puis bleue.

-De faire demi-tour si les couleurs des lignes détectées sont verte puis noire, noire puis bleue, blanche puis verte ou bleue puis blanche.

- De pivoter de 90° vers la droite si les couleurs détectées sont rouge puis noire ou noire puis jaune.

-De pivoter de 90° vers la gauche si les couleurs détectées sont jaune puis noire ou noire puis rouge.

Le test se résume à vérifier le bon repositionnement du robot dans le terrain et de vérifier qu'une fois bien placé, l'angle courant est bien nul, c'est à dire aligné avec sur en but.

```
replacement();
System.out.println("fin, angleCrt= "+angleCrt);
Button.waitForAnyPress();
```

Le test se résume à vérifier le bon repositionnement du robot dans le terrain et de vérifier qu'une fois bien placé, l'angle courant est bien nul, c'est à dire placé face à l'en-but adverse.