MAL

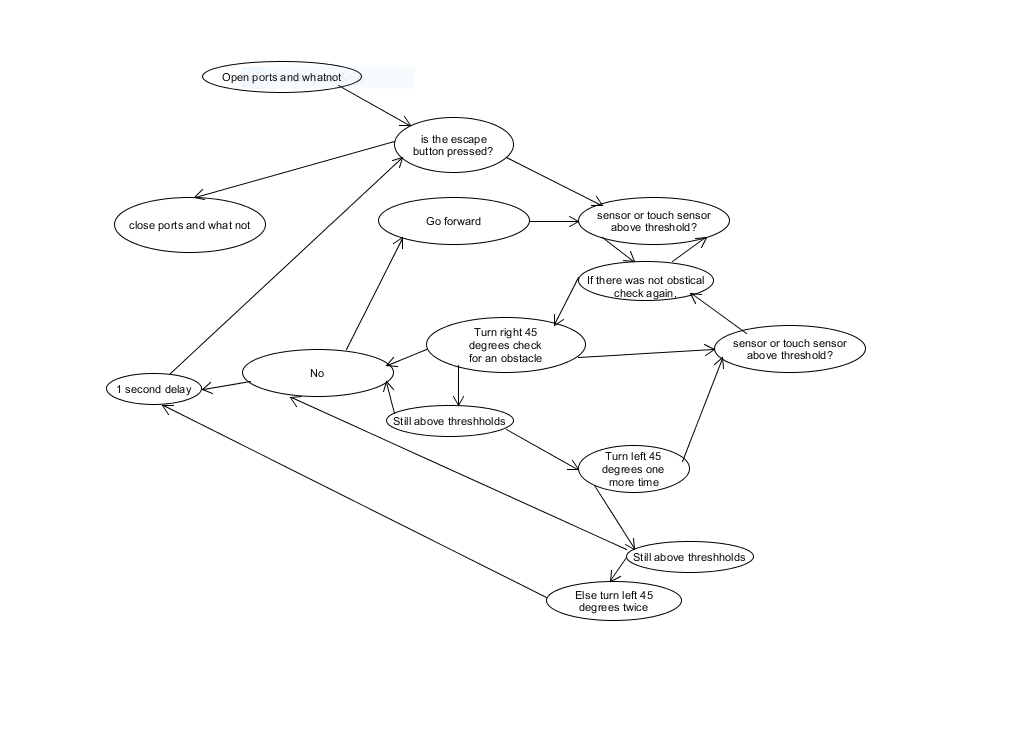
Mark Garrett

Section 1, design summary

In terms of hardware I designed Mal around the treads. I was course how one would go about using them. So I asked myself how someone would use them. I started attaching the gears that went with the trends to a framework. So then the question was how to power it, so I looked at different ways to mount the motors to the treads. This proved difficult, so I used the legs to connect the treads. In the end I decided to put one motor in the front and another in the back. I mounted the brick to motors and the Lego skeleton connecting the treads. I originally I had mounted the ultra-sonic sensor to the swivel, but I could not find a way to have the ultra-sonic sensor be able to look left and right, and still be low to the ground. I decided having a low ultra-sonic sensor low to the ground would be very important, because some obstacles might be too short if I mounted where I original mounted it. If the cable port on the ultra-sonic sensor had been in a different spot it might have worked. Having a fixed ultra-sonic sensor meant Mal had blind spots, and would get stuck on walls at 45 degree angles. I fixed this by adding a bumper attracted to touch sensors.

In terms of software I stared with some pseudocode with each line written like a java method. Then I implemented each of the methods. Whenever it moves forward it makes consecutive calls to my isObstical() method which compares the current state of both the touch sensors and the ultra-sonic sensor. See pseudocode for the algorithm. Turning was a bit weird because one motor is mounted backward. I turn by running the treads in opposite directions.

Section 2, code graph and pseudocode



Pseudocode

Open ports and whatnot

Run while the escape button is not pressed

Go forward until the is an obstacle

Turn right 45 degrees check for an obstacle

If no obstacle, go forward

Else turn left 45 degrees twice

Turn left 45 degrees one more time (I thought this would mean a dead end in a maze)

1 second delay

Back to the beginning of the loop if the

If the escape button was pressed close ports and whatnot

Section 4, Design improvement

Using wheels and utilizing gear ratios would have been smart for improving the time. In terms of the algorithm, I should have accommodated the 45 degree turns. In my original algorithm. I used 90 degree turns. 90 degree turns caused Mal to get turned around so much that it was facing the wrong way. It also would not have been able to navigate a course that had both left and right turns, and was just as spaces as the course in the project lab. If I kept track of positional data somehow, that could also help the design.

Section 5, Java (also in attached file)

import lejos.hardware.Button;

import lejos.hardware.Sound;

import lejos.hardware.ev3.LocalEV3;

import lejos.hardware.motor.Motor;

import lejos.hardware.port.Port;

import lejos.hardware.sensor.EV3TouchSensor;

import lejos.hardware.sensor.EV3UltrasonicSensor;

import lejos.hardware.sensor.SensorMode;

import lejos.robotics.RegulatedMotor;

import lejos.utility.Delay;

import lejos.hardware.lcd.LCD;

public class mal {

public static final int knockBack = 1250;

public static final int turnTime = 1000;

public static final double obsticalTolerance = .2;

public static final double collisionTolerance = 0;

public static RegulatedMotor[] syncList;

public static final int speed = 900;

public static final int backSpeed = 600;

public static final int turnSpeed = 150;

public static SensorMode distancefinder;

public static EV3UltrasonicSensor ultra;

public static EV3TouchSensor leftBumper;

public static float[] leftTsample;

public static EV3TouchSensor rightBumper;

public static float[] rightTsample;

public static SensorMode touchproviderRight;

public static SensorMode touchproviderLeft;

public static final int antiDrift = 22;

public static void setup(){

syncList = new RegulatedMotor[1];

syncList[0] = Motor.D;

Port SonicPort = LocalEV3.get().getPort("S4");

ultra = new EV3UltrasonicSensor(SonicPort);

Port leftPort = LocalEV3.get().getPort("S3");

leftBumper = new EV3TouchSensor(leftPort);

touchproviderLeft = (SensorMode) leftBumper.getTouchMode();

leftTsample = new float[leftBumper.sampleSize()];

Port rightPort = LocalEV3.get().getPort("S2");

rightBumper = new EV3TouchSensor(rightPort);

touchproviderRight = (SensorMode) rightBumper.getTouchMode();

rightTsample = new float[rightBumper.sampleSize()];

distancefinder = (SensorMode) ultra.getDistanceMode();

Sound.beep();

}

public static void main(String[] args){

setup();

while(isNotEscapePressed()){

goForward();

if(isNotEscapePressed()){

stepBack();

lookRight();

}

if(isObstical()&&isNotEscapePressed()){

lookLeft();

lookLeft();

}

if(isObstical()&&isNotEscapePressed()){

lookLeft();

}

if(isNotEscapePressed()){

Delay.msDelay(1000);

}

}

shutdown();

}

public static void goForward(){

LCD.drawString("goForward", 0, 0);

Motor.D.setSpeed(speed);

Motor.C.setSpeed(speed+antiDrift);

Motor.C.synchronizeWith(syncList);

Motor.C.startSynchronization();

Motor.D.forward();

Motor.C.backward();

Motor.C.endSynchronization();

//int zig = 0;

while(!isObstical()&&isNotEscapePressed()){

// zig++;

// if(zig>11)

// zig=-5;

// Motor.D.setSpeed(speed+zig);

// Motor.C.setSpeed(speed+zig+antiDrift);

}

Motor.C.synchronizeWith(syncList);

Motor.C.startSynchronization();

Motor.D.stop();

Motor.C.stop();

Motor.C.endSynchronization();

}

public static void lookLeft(){

LCD.drawString("LookLeft", 0, 0);

Motor.D.setSpeed(turnSpeed);

Motor.C.setSpeed(turnSpeed+antiDrift);

Motor.C.synchronizeWith(syncList);

Motor.C.startSynchronization();

Motor.D.forward();

Motor.C.forward();

Motor.C.endSynchronization();

Delay.msDelay(turnTime);

Motor.C.synchronizeWith(syncList);

Motor.C.startSynchronization();

Motor.D.stop();

Motor.C.stop();

Motor.C.endSynchronization();

}

public static void lookRight(){

LCD.drawString("lookRight", 0, 0);

Motor.D.setSpeed(turnSpeed);

Motor.C.setSpeed(turnSpeed+antiDrift);

Motor.C.synchronizeWith(syncList);

Motor.C.startSynchronization();

Motor.D.backward();

Motor.C.backward();

Motor.C.endSynchronization();

Delay.msDelay(turnTime);

Motor.C.synchronizeWith(syncList);

Motor.C.startSynchronization();

Motor.D.stop();

Motor.C.stop();

Motor.C.endSynchronization();

}

public static boolean isObstical(){

touchproviderLeft.fetchSample(leftTsample, 0);

touchproviderRight.fetchSample(rightTsample, 0);

LCD.drawString("isObstical", 0, 0);

float[] sample = new float[ultra.sampleSize()];

distancefinder.fetchSample(sample, 0);

return sample[0]<obsticalTolerance || leftTsample[0]>collisionTolerance || rightTsample[0]>collisionTolerance;

}

public static void stepBack(){

LCD.drawString("stepBack", 0, 0);

Motor.D.setSpeed(backSpeed);

Motor.C.setSpeed(backSpeed+antiDrift);

Motor.C.synchronizeWith(syncList);

Motor.C.startSynchronization();

Motor.D.backward();

Motor.C.forward();

Motor.C.endSynchronization();

Delay.msDelay(knockBack);

Motor.C.synchronizeWith(syncList);

Motor.C.startSynchronization();

Motor.D.stop();

Motor.C.stop();

Motor.C.endSynchronization();

}

public static void shutdown(){

LCD.drawString("shutdown", 0, 0);

ultra.close();

leftBumper.close();

rightBumper.close();

}

public static boolean isNotEscapePressed(){

return Button.getButtons()!=Button.ID\_ESCAPE;

}

}