**Geometric Pattern Program: Step by Step Instructions**

Here is a review of the steps you could take to program your robot to drive in a geometric pattern. It assumes that you have built a “differential drive” robot with two motors, connected to port A and D, respectively (see the construction blueprints available at our home page).

**Step 1:** Make sure your laptop communicates with your brick: Either plug it in with the USB cable or - much preferred - click on the Bluetooth icon, pick "Join a Personal Area Network", and wait for your brick to appear. Right-click on its icon and select "Connect using | Access Point". Of course make sure your brick is turned on.

**Step 2:** Start a 'clean' new project: Start Eclipse and select "File | New | Project", then highlight "LeJOS Project" and click "Next". Type in a name for your project – perhaps something like “Geometry”, double check that the entry for “Use an execution environment JRE” says “Java-SE 1.7”, and click “Finish” to create the project. Finally, highlight the project you just created and select “File | New | Class” and name it also “Geometry”. Make sure you check the box to create the standard “main” method, as usual, then click “Finish”.

**Step 3:** Add two fields named leftMotor and rightMotor for the motors connected to output ports A and D, respectively

**public** **class** Geometry

{

 **public** **static** EV3LargeRegulatedMotor *leftMotor* =

 **new** EV3LargeRegulatedMotor(MotorPort.***A***);

 **public** **static** EV3LargeRegulatedMotor *rightMotor* =

 **new** EV3LargeRegulatedMotor(MotorPort.***D***);

 **public** **static** **void** main(String[] args)

 {

 }

}

This will generate some errors because neither EV3LargeRegulatedMotor nor MotorPort is known to the compiler. Click on the “lightbulb” icon on the left side and choose the “Import” statement to fix the errors. This will place two “import” statements at the top of your source code and you should have no more errors.

**Step 4:** Now that you added the parts your robot “has”, namely the fields, you need to add the things your robot should “do”, in other words the methods. You want your robot to drive forward by a given amount, and to turn by a given angle. Thus, you need to add two methods to your code, both with one input parameter, as follows.

**import** lejos.hardware.motor.EV3LargeRegulatedMotor;

**import** lejos.hardware.port.MotorPort;

**public** **class** Geometry

{

 **public** **static** EV3LargeRegulatedMotor *leftMotor* =

 **new** EV3LargeRegulatedMotor(MotorPort.***A***);

 **public** **static** EV3LargeRegulatedMotor *rightMotor* =

 **new** EV3LargeRegulatedMotor(MotorPort.***A***);

 // method to drive forward by 'distance' centimeters

 **public** **static** **void** drive(**int** distance)

 {

 }

 // method to turn by 'angle' degrees

 **public** **static** **void** turn(**int** angle)

 {

 }

 **public** **static** **void** main(String[] args)

 {

 }

}

Note the use of comments (the plain English lines preceded by two slashes, shown in green). Comments are useful to explain your code but they don’t have any effect on your program. So, right now your methods don’t actually do anything at all. But you should not have any errors, at least, and since you have figured out which methods you need and what the method headers are, the rest of the program is relatively easy.

**Step 5:** Figure out the turn method. The input to the method should be the number of degrees that your robot should turn. As discussed in class, we want to rotate one motor forward and the other one backwards to turn in place. But the question is by how much, since rotating the *wheels* by, say 90 degrees will most likely *not* correspond to a 90 degree rotation of your *robot*. So, you need to figure out a *conversion factor* which you can determine through trial and error: start with an arbitrary decimal number, say 5.4. Multiply the input angle by that number, convert the result back to an integer (see below for the exact syntax), and store the result in a new variable of type int named rotDegree:

 // method to turn by 'angle' degrees

 **public** **static** **void** turn(**int** angle)

 {

 Int rotDegree = (int)(5.4\*angle);

 leftMotor.rotate(rotDegree, true);

rightMotor.rotate(rotDegree, true);

 }

This might work perfectly, or your robot might turn too much, or it might turn too little.

**Step 6:** Now comes the experimentation part: add a call to this turn method in your main method to turn by 180 degrees:

 **public** static **void** main(String[] args)

 {

 turn(180);

 }

Execute this program and check what happens:

* if your robot turns *more* than 180 degrees, *decrease* the factor in the turn method
* if your robot turns *less* than 180 degrees, *increase* the factor in the turn method

Keep changing the factor until your robot turns a perfect 180 degrees. Once you have determined the right factor, the robot will turn correctly for any other input as well, so turn(90) will now result in a 90 degree turn, turn(120) in a 120 degree turn, etc.

**Step 7:** Implement the drive method in much the same way. Pick an arbitrary factor, compute a new rotDegree variable inside the drive method, and adjust it so that the call in the main method to drive 10 cm really makes your robot drive forward by 10 cm. Experiment until you find the right factor for this method (which will be different from the factor in the turn method).

**Step 8:** Now you can add a sequence of drive and turn directives in the main method to make your robot go in the desired geometric shape. For example, if you call turn(90) and drive(50) four times, your robot should execute a square of side length 50 cm.