**Ant Simulator**

**Goal**: Create a robot that keeps track of its position and heading, similar to an ant. The robot should randomly explore its surroundings, then move back to its nest in a straight line.

**Step 1**: Creating a new program (aka class) named “AntSimulator”. Copy and paste the code below into your new class. It will be the foundation of our Ant simulation program, containing for now a turn and travel method that maintain proper heading and current location. We discussed that before the break (see last PP sides) so hopefully the code will make sense.

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

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

**public** **class** AntSimulator

{

// constants depending on the physical parameters of our robot

**public** **static** **final** **double** ***TURN\_FACTOR*** = 2.17; // needs calibration

**public** **static** **final** **double** ***TRAVEL\_FACTOR*** = 15.5; // needs calibration

**public** **static** **final** **int** ***TURN\_SPEED*** = 150;

**public** **static** **final** **int** ***TRAVEL\_SPEED*** = 400;

// fields

**public** **static** EV3LargeRegulatedMotor *motorRight* =

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

**public** **static** EV3LargeRegulatedMotor *motorLeft* =

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

// fields to store current location (x,y) and heading

**public** **static** **double** *x* = 0.0;

**public** **static** **double** *y* = 0.0;

**public** **static** **double** *heading* = 0.0;

// method to turn robot counter-clockwise (if angle > 0) and update its heading

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

{

*motorLeft*.setSpeed(***TURN\_SPEED***);

*motorRight*.setSpeed(***TURN\_SPEED***);

**int** degrees = (**int**)(***TURN\_FACTOR*** \* angle);

*motorLeft*.rotate(degrees, **true**);

*motorRight*.rotate(-degrees);

*heading* = (*heading* + angle) % 360;

}

// method to travel forward in a straight line by dist cm and updating its (x,y)

// location

**public** **static** **void** travel(**double** dist)

{

*motorLeft*.setSpeed(***TRAVEL\_SPEED***);

*motorRight*.setSpeed(***TRAVEL\_SPEED***);

**int** degrees = (**int**)(***TRAVEL\_FACTOR*** \* dist);

*motorLeft*.rotate(degrees, **true**);

*motorRight*.rotate(degrees);

*x* = *x* + dist\*Math.*cos*(*heading* / 180.0 \* Math.***PI***);

*y* = *y* + dist\*Math.*sin*(*heading* / 180.0 \* Math.***PI***);

}

// standard main method

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

{

}

}

**Step 2:** Next we need to calibrate the constants to ensure that turn and travel work correctly. Add a call to “turn(180)” to the main method and execute. If robot turns clockwise, adjust negative sign. If robot turns less than 180 degrees, increase TURN\_FACTOR. If robot turns more than 180 degrees, decrease TURN\_FACTOR. Keep on going until your robot makes a perfect 180 degree turn counter-clockwise.

Replace the call to “turn(180)” by a call to “travel(40)” and execute. If the robot turns backwards, adjust the way the motors turn. If the robot drives more than 40 cm ahead, decrease TRAVEL\_FACTOR. If the robot drives less than 40 cm, decrease TRAVEL\_FACTOR. Keep on going until your robot travels forward by 40 cm perfectly.

*Note:* To make the movement less jerky, set the acceleration of both motors to, say, 800, at the beginning of “main”.

**Step 3**: Verify that the robot maintains its location properly. To do that, make the robot move in a square of side length 50 and print out the location and heading of the robot when done. For practice, use a ‘counting loop” for this. What should it say? Does it?

**Step 4:** Add a method driveTo(x,y) that drives the robot from its current location to a particular (x, y) spot in our imagined coordinate system. Add additional methods as needed.

We do need two additional methods: findDistanceBetween( (x1,y1) , (x2, y2) ) and findAngleBetween( (x1,y1), (x2,y2)). Refer to the last PP slides to see wy hese methods work:

// Finding the distance between points (x1,y1) and (x2,y2)

**public** **static** **double** findDistanceBetween(**double** x1,**double** y1, **double** x2,**double** y2)

{

**return** Math.*sqrt*((x2-x1)\*(x2-x1) + (y2-y1)\*(y2-y1));

}

// Finding the angle in degrees between points (x1,y1) and (x2,y2)

**public** **static** **double** findAngleBetween(**double** x1,**double** y1, **double** x2,**double** y2)

{

**return** Math.*atan2*(y2-y1, x2-x1) \* 180 / Math.***PI***;

}

With these functions defined, the method driveTo(x, y) is relatively simple. Note that it needs to take the current heading into account to figure out how much to turn to point towards (x2, y2):

**public** **static** **void** driveTo(**double** xNew, **double** yNew)

{

**double** angle = *findAngleBetween*(*x*,*y*, xNew,yNew);

**double** dist = *findDistanceBetween*(*x*,*y*, xNew,yNew);

*turn*(angle - *heading*);

*travel*(dist);

}

**Step 5:** Test the new driveTo(x,y) method by driving in a rectangular pattern, specifying each corner by a driveTo command.

If you are satisfied that driveTo works properly, we can complete phase 1 of our ant challenge: create a counting loop to drive to, say, 6 random coordinates (x, y), where 0 < x,y < 100. When done, return to the origin and turn like you were at the beginning.

The complete code so far is as follows. It should solve the first part of the “ant challenge”. Note that the constants may not work for you; you will need to calibrate them (see step 2 above) for your particular robot.

**public** **class** AntSimulator

{

// constants depending on the physical parameters of our robot

**public** **static** **final** **int** ***ACCEL*** = 800;

**public** **static** **final** **double** ***TURN\_FACTOR*** = 2.17; // needs calibration

**public** **static** **final** **double** ***TRAVEL\_FACTOR*** = 18.5; // needs calibration

**public** **static** **final** **int** ***TURN\_SPEED*** = 150;

**public** **static** **final** **int** ***TRAVEL\_SPEED*** = 400;

// fields

**public** **static** EV3LargeRegulatedMotor *motorRight* =

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

**public** **static** EV3LargeRegulatedMotor *motorLeft* =

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

// fields to store current location (x,y) and heading

**public** **static** **double** *x* = 0.0;

**public** **static** **double** *y* = 0.0;

**public** **static** **double** *heading* = 0.0;

// method to turn robot counter-clockwise (if angle > 0) and update its heading

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

{

*motorLeft*.setSpeed(***TURN\_SPEED***);

*motorRight*.setSpeed(***TURN\_SPEED***);

**int** degrees = (**int**)(***TURN\_FACTOR*** \* angle);

*motorLeft*.rotate(degrees, **true**);

*motorRight*.rotate(-degrees);

*heading* = (*heading* + angle) % 360;

}

// method to travel forward by dist cm and updating its (x,y) location

**public** **static** **void** travel(**double** dist)

{

*motorLeft*.setSpeed(***TRAVEL\_SPEED***);

*motorRight*.setSpeed(***TRAVEL\_SPEED***);

**int** degrees = (**int**)(***TRAVEL\_FACTOR*** \* dist);

*motorLeft*.rotate(degrees, **true**);

*motorRight*.rotate(degrees);

*x* = *x* + dist\*Math.*cos*(*heading* / 180.0 \* Math.***PI***);

*y* = *y* + dist\*Math.*sin*(*heading* / 180.0 \* Math.***PI***);

}

// finding the distance between points (x1,y1) and (x2,y2)

**public** **static** **double** findDistanceBetween(**double** x1,**double** y1,**double** x2,**double** y2)

{

**return** Math.*sqrt*((x2-x1)\*(x2-x1) + (y2-y1)\*(y2-y1));

}

// finding the angle in degrees between points (x1,y1) and (x2,y2)

**public** **static** **double** findAngleBetween(**double** x1, **double** y1, **double** x2, **double** y2)

{

**return** Math.*atan2*(y2-y1, x2-x1) \* 180 / Math.***PI***;

}

// drives to a given (x,y) location

**public** **static** **void** driveTo(**double** xNew, **double** yNew)

{

**double** angle = *findAngleBetween*(*x*,*y*, xNew,yNew);

**double** dist = *findDistanceBetween*(*x*,*y*, xNew,yNew);

*turn*((angle – *heading) % 360*);

*travel*(dist);

}

// standard main method

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

{

*motorLeft*.setAcceleration(***ACCEL***);

*motorRight*.setAcceleration(***ACCEL***);

**int** count = 0;

**while** (count < 4)

{

*driveTo*( 130\*Math.*random*(), 80\*Math.*random*());

count++;

}

Sound.buzz();

*driveTo*(0,0);

*turn*(-*heading*);

}

}

**Step 6**: If we want to locate food represented by a green dot, we need to attach a downward facing color sensor to the front of our robot. Let’s say we connect that sensor to sensor port 1. We can then add the appropriate framework for a sensor to our code and add a detectFood method that returns true as soon as the color sensor has found something green.

Add three fields to our AntSimulator as follows:

// Get an instance of a color sensor in color detection mode

**private** **static** EV3ColorSensor *colorSensor* = **new** EV3ColorSensor(SensorPort.***S1***);

**private** **static** SampleProvider *colorProvider* = *colorSensor*.getColorIDMode();

**private** **static** **float**[] *colorData* = **new** **float**[*colorProvider*.sampleSize()];

Now we can add a foodFound method as follows:

// check the color sensor and return true if it detects anything green

**public** **static** **boolean** foundFood()

{

*colorProvider*.fetchSample(*colorData*, 0);

**if** (*colorData*[0] == Color.***GREEN***)

{

Sound.*beep*();

**return** **true**;

}

**else**

{

**return** **false**;

}

}

It now seems relatively easy to change the main method so that our ant robot searches at random until it finds any food:

**while** (!*foodFound*())

{

*driveTo*( 130\*Math.*random*(), 80\*Math.*random*());

}

Note that an exclamation point in front of a boolean variable negates it. So, you should read ! as “not”. In other words !foodDetected() reads like “not foodDetected”.

But this code does not work, or more precisely it works only in special cases not in general. Why?

The problem, of course, is that the method driveTo will drive the robot to the new location *without checking for food along the way*! Neither the turn nor the travel method stop before they have completed the various rotations. Thus, the above loop checks for food only when a driveTo command is over and just before the next one starts.

To fix this, we need to modify the turn and the travel method so that we can insert a check for food while the turn (or travel) is still ongoing. Let’s look at travel:

**public** **static** **void** travel(**double** dist)

{

*motorLeft*.setSpeed(***TRAVEL\_SPEED***);

*motorRight*.setSpeed(***TRAVEL\_SPEED***);

**int** degrees = (**int**)(***TRAVEL\_FACTOR*** \* dist);

*motorLeft*.rotate(degrees, **true**);

*motorRight*.rotate(degrees);

*x* = *x* + dist\*Math.*cos*(*heading* / 180.0 \* Math.***PI***);

*y* = *y* + dist\*Math.*sin*(*heading* / 180.0 \* Math.***PI***);

}

One approach is to add “true” to the call to motorRight.rotate, which cause it to instruct the motor to rotate but execute the next command immediately, while leaving it to the motor itself to check and finish after rotating the required amount. But then that method would be over before the rotation is complete, so the robot would not drive the required distance at all. To prevent that, we add a loop that keeps checking if the motor is still rotating, while at the same time looking for food. Here is the new code:

**public** **static** **void** travel(**double** dist)

{

*motorLeft*.setSpeed(***TRAVEL\_SPEED***);

*motorRight*.setSpeed(***TRAVEL\_SPEED***);

**int** degrees = (**int**)(***TRAVEL\_FACTOR*** \* dist);

*motorLeft*.rotate(degrees, **true**);

*motorRight*.rotate(degrees, **true**);

**while** ((*motorLeft*.isMoving()) && (!*foundFood*()))

{

}

*x* = *x* + dist\*Math.*cos*(*heading* / 180.0 \* Math.***PI***);

*y* = *y* + dist\*Math.*sin*(*heading* / 180.0 \* Math.***PI***);

}

So, if we consider the code so far:

* Our “ant” can drive to random locations while keeping track of its position and heading
* If we drive to a fixed number of random locations, say using a counting loop, we can return back to where we started
* If we change the code to drive to random locations until we locate some food, our robot correctly detects the food, but it will not return to its nest any longer – something is off – any suggestions?

We will fix that on Wed. As HW for Wed, create a program that solves part 1 of our ant challenge, i.e. an ant that drives to 6 (say) random locations, then returns home. We will deal with finding food in class on Wed.