**Ant Simulator (Final Version)**

So far we have created an ant simulator program to solve part 1 of our ant challenge. Here is the code:

**public** **class** AntSimulator1

{

 // 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*);

 }

}

We also made progress creating a program to solve part 2 of the ant challenge, but it did not quite work. Here is where we left off: we started with the above code and:

* We added the standard framework to integrate a color sensor into our program, i.e. we added three fields and one new method called foodFound()
* Of course we also added a color sensor in front of our robot, facing down, connected to sensor port 1
* Then we changed the main method so that our ant robot searches at random until it finds any food
* Finally, we modified the travel method to stop as soon as we find something green

Here is the new code. Make sure to create a new class named AntSimulator2 and “copy and paste” the following code

**public** **class** AntSimulator2

{

 // 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***);

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

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

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

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

 // 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;

 // 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**;

 }

 }

 // 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;

 }

 **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***);

 }

 // 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***);

 **while** (!*foundFood*())

 {

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

 }

 Sound.*buzz*();

 *driveTo*(0,0);

 *turn*(-*heading*);

 }

}

Compile it and execute to see what is happening and what you still have to fix.

**Problem 1**: The first problem is in the method travel. It now might stop at any time (when we find some food). But then the distance traveled is no longer dist, which is used to update (x,y). Thus, the coordinate system will no longer agree with where the robot really is.

**Solution 1**: Compute the actual distance travelled, using the tachometers integrated in the motors. In other words, recompute dist to stand for the *actual* distance traveled before using it to update x and y:

dist = (*motorLeft*.getTachoCount() + *motorRight*.getTachoCount()) / (2.0\****TRAVEL\_FACTOR***);

Note that you also need to make sure the tachometers for both motors are reset to zero when the method starts.

**Problem 2**: This problem is more subtle. When we check foundFood() we want to make sure to check it only while we are searching for food. Once we found food, we can abandon the search and instead run home.

**Solution 2:** To accomplish that we use two boolean variables that act as switches that determine what the program is supposed to do at any given time. So, define foodFound and runningHome as two boolean fields, both initialized to false. If foundFood is true, the travel method will stop immediately, while runningHome dictates whether we still need to search for food. To summarize, we change the code of travel as follows:

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

 {

 *motorLeft*.resetTachoCount();

 *motorRight*.resetTachoCount();

 *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*))

 {

 **if** (!*runningHome*)

 *foundFood* = *foundFood*();

 }

 // Manually stopping motors in case we found food

 *motorLeft*.stop(**true**);

 *motorRight*.stop();

 // the travel method could quite before it has traveled the input distance (when it

 // finds food somewhere along the way). Thus, we recompute 'dist' to stand for the

 // actual distance traveled.

 dist = (*motorLeft*.getTachoCount()+*motorRight*.getTachoCount())/(2.0\****TRAVEL\_FACTOR***);

 // Now we can use the actual distance traveled to adjust x and y coordinates

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

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

 }

Finally change the method foundFood to set runningHome to true after finding something green:

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

 {

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

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

 {

 Sound.*beep*();

 *runningHome* = **true**;

 **return** **true**;

 }

 **else**

 {

 **return** **false**;

 }

 }

Our last change involves the main method:

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

 {

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

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

 *foundFood* = **false**;

 *runningHome* = **false**;

 **while** (!*foundFood*)

 {

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

 }

 *foundFood* = **false**; // so that the travel method executes properly

 *runningHome* = **true**; // so that we don't check for food again

 Sound.*buzz*();

 *driveTo*(0,0);

 *turn*(-*heading*);

 }

Finally we want to implement different search patterns instead of searching at random. We define a search pattern in a list called path, each entry in the list consisting of two numbers, an x and a y coordinate. For example:

double[][] path = { {130, 80}, {0,80}, {130,0}, {0,0}};

To use this path, modify the loop in the main method to the following:

double[][] path = { {130, 80}, {0,80}, {130,0}, {0,0}};

**int** count = 0;

**while** ((!*foundFood*) && (count < *PATH*.length))

{

 *driveTo*(*PATH*[count][0], *PATH*[count][1]);

 count++;

}

Come up with your own search pattern for finding the food. The pattern that finds the food the fastest will get extra points. Here are a few possible paths:

**Back and Forth**

**double** [][] path = {{120, 0}, {0, 10},

 {120, 10}, {0, 20},

 {120, 20}, {0, 30},

 {120, 30}, {0, 40},

 {120, 40}, {0, 50},

 {120, 50}, {0, 60},

 {120, 60}, {0, 70},

 {120, 70}, {0, 80},

 {120, 80}};

**A Spiral**

**double** [][] path = {{120, 0 }, {120, 80}, { 0, 80), { 0, 10},

 {110, 10}, {110, 70}, {10, 70}, {10, 20},

 {100, 20}, {100, 60}, {20, 60}, {20, 30},

 { 90, 30}, { 90, 50}, {30, 50}, {30, 40},

 { 80, 40}, { 80, 40}, {40, 40}, {40, 50}};

**Random Height Back and forth**

**double** [][] path = {{120, 80\*Math.*random*()}, {0, 80\*Math.*random*()},

 {120, 80\*Math.*random*()}, {0, 80\*Math.*random*()},

 {120, 80\*Math.*random*()}, {0, 80\*Math.*random*()},

 {120, 80\*Math.*random*()}, {0, 80\*Math.*random*()},

 {120, 80\*Math.*random*()}, {0, 80\*Math.*random*()},

 {120, 80\*Math.*random*()}, {0, 80\*Math.*random*()},

 {120, 80\*Math.*random*()}, {0, 80\*Math.*random*()},

 {120, 80\*Math.*random*()}, {0, 80\*Math.*random*()},

 {120, 80\*Math.*random*()}, {0, 80\*Math.*random*()},

 {120, 80\*Math.*random*()}, {0, 80\*Math.*random*()};