**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.