Patterns, frameworks & Lejos

part 2: Navigation
Navigation & the Navigator API

• Dead reckoning: navigation method based on direction & distance (used by pirates & other seafarers for eons)

• In traditional dead reckoning, direction is obtained from a magnetic compass, while distance might be measured in any number of ways (knowing time & speed, distance is easy to calculate)

• Lego robots don’t come equipped with compasses, but there are ways to work around this
A little trig

- Trigonometry, or the study of triangles, comes in handy when you’re trying to navigate with dead reckoning.
- To find a robot’s current position, given the distance traveled from the origin and the adjacent angle, the formula is:
  - \( y = \cos(\text{angle}) \times \text{distance} \)
  - \( x = \sin(\text{angle}) \times \text{distance} \)
  - In Java, that would be:
    ```java
double x = Math.cos(Math.toRadians(angle)) \* distance;
double y = Math.sin(Math.toRadians(angle)) \* distance;
```
The Navigator API

- Provides set of methods for controlling robot movement
- Includes methods for:
  - moving to a specific location
  - controlling direction
- Navigator object keeps track of current angle & x,y coordinates as each movement is performed
- Can be used by any robot with differential steering that can turn within its own footprint (not changing its coordinates)
Navigator interface methods

- public void forward()
- public void backward()
  - move robot in specified direction until stop() is called
- public void travel(int distance)
  - moves specified distance; positive means forward, negative backward
  - distance is in centimeters;
- public void stop()
  - halts robot & calculates new x,y coordinates
Navigator interface methods

• public void rotate(float angle)
  – angle unit is degrees
  – may be positive or negative
• public void gotoAngle(float angle)
  – rotates robot to point in specified direction
  – takes shortest path to reach specified angle, which may be positive or negative
• public void gotoPoint(float x, float y)
  – move robot to specified x,y coordinates
  – if stop() method called during movement, robot stops and new x,y coordinates are calculated
Navigator interface methods

- public float getX(): returns current x coordinate
- public float getY(): returns current y coordinate
- public float getAngle(): returns current angle robot is facing; should not be called while robot is rotating
Implementing classes

• There are two classes in the Lejos API that implement the Navigator interface:
  – TimingNavigator: keeps track of coordinates by measuring movement in terms of time
  – RotationNavigator: assumes the use of two rotation sensors (one for each motor)
TimingNavigator constructor

• As with all constructors, allocates and object and initializes key variables; for TimingNavigator, parameters include:
  – controller outputs for the left and right wheels (A and C, for example)
  – the number of seconds it takes the robot to drive 100 centimeters (about 2.0 seconds for a fast robot, 5.0 for a slow one)
  – the number of seconds it takes the robot to rotate 360 degrees (depends on axle length; .646 is normal for short axle, 2.2 for long)

• Sets \( x, y, \) and starting angle values to 0
The setMomentumDelay method

• Used to add extra time to each rotation, giving robot time to overcome its momentum before rotating; increases accuracy

• Time argument is given in milliseconds
TimingNavigator robot characteristics

• For reasonably accurate navigation, your robot should:
  – be relatively slow so that wheels don’t skid when stopping
  – rotate slowly (again, to avoid skidding)
  – start immediately when instructed to move

• A robot with tank treads is particularly suited to the TimingNavigator idea
import josx.platform.rcx.*;
import josx.robotics.*;

public class Trilobot {
    public static void main (String [] args) {
        Motor.A.setPower(7);
        Motor.C.setPower(5);

        TimingNavigator n = new TimingNavigator(Motor.C,
                                                Motor.A, 5.475f, 4.03f);
        n.rotate(360);
        n.gotoPoint(100,0);
        n.gotoPoint(100,100);
        n.gotoPoint(0,100);
        n.gotoPoint(0,0);
    }
}
TimingNavigator accuracy and sources of errors

• The accuracy of a TimingNavigator robot is far from perfect; at best it roughly follows a prescribed course

• Possible sources of errors include:
  – systematic errors (built into the robot design and/or software parameter values)
  – non-systematic errors: introduced by environmental factors (uneven surface, e.g.)
Systematic errors

• Accuracy of travel and rotate times: any inaccuracy in initial values will be multiplied over time as robot moves

• Underrotation: robot may fall short of discrete angle because it must overcome its tendency to remain at rest (see Newton’s first law of motion) – the setMomentumDelay method helps to overcome this problem

• Battery power varies, and the batteries slowly lose power as long as the robot is on
Nonsystematic errors

- Carpet and other rough surfaces produce more inaccuracies than smooth surfaces
- Contact with other objects may throw the robot out of alignment – remember, touch sensors report collisions only after they happen
Navigation with rotation sensors

• Rotation sensor is used to count the number of rotations of an axle
• Adds one for every 22.5 degrees rotated:
  – a full 360-degree rotation adds 16 to the count
  – the RCX uses a 16-bit integer, so the maximum count value is 32767 (adding one will roll over to -32768)
  – reversing direction causes counter to decrement
Rotation sensor accuracy

• Sensor has no memory; RCX has to keep track of running total
• Lejos can accurately count intervals at a rate of 9600 per minute (600 RPM or 10 RPS)
  – Ungeared Lego motor spins at about 350 RPM
  – Gear reduction may affect accuracy
Using a trailer odometer

• To get distance in meters, we must convert the number of “ticks” on the rotation sensor to the distance per wheel rotation

• The measurement is based on:
  – the size of the wheel to which the rotation sensor is attached
  – the number of ticks per complete rotation (already known to be 16)
Calculating distance

• The tires in the RCX kit are labeled with their diameters (in mm)
• For example, the large “motorcycle” tires are 81.6 mm, or 8.16 cm
• Therefore the circumference of the wheel is $8.16\pi$, or 25.64 cm; so $1/16^{th}$ of a rotation (one tick) measures a distance of 1.6025 cm
Measurer code

```java
import josx.platform.rcx.*;

public class Measurer implements ButtonListener, SensorConstants {
    private final float WHEEL = 8.16f * (float)Math.PI;
    private final float INTERVAL = WHEEL / 16;

    public Measurer() {
        Sensor.S2.setTypeAndMode(SENSOR_TYPE_ROT,
                                   SENSOR_MODE_ANGLE);
        Sensor.S2.setPreviousValue(0);
        Sensor.S2.activate();
        Button.PRGM.addButtonListener(this);
    }
}
```
public static void main (String [] args) {
    new Measurer().measure();
}

public void measure () {
    while (true) {
        int ct = Sensor.S2.readValue();
        int cm = (int)(ct * INTERVAL);
        LCD.setNumber(0x3001, cm, 0x3004);
        LCD.refresh();
    }
}

public void buttonPressed(Button b) {
    Sensor.S2.setPreviousValue(0);
    Sound.beepSequence();
}

public void buttonReleased(Button b) {}
Odometer placement and distance calculation

• An on-axle odometer (such as the one used on our pre-built trailer) gives a 1:1 ratio between wheel and sensor rotation.

• If the robot uses differential drive steering, rotation sensors can be places on each axle to measure distance driven by each wheel, increasing accuracy.

• The odometer can also be place off-axle, using gears to translate movement; placing a smaller gear on the rotation sensor axle means it turns more times than the wheel axle, giving finer readings.
The RotationNavigator Class

- Implements the Navigator interface, and so is similar to the TimingNavigator class
- Assumes:
  - two rotation sensors (one for each wheel or axle)
  - forward motion of motor = forward motion of robot
  - rotation sensor records positive movement when robot moves forward
RotationNavigator constructor

- Two versions: long form requires arguments for:
  - wheel diameter: drive wheel diameters (float)
  - drive length: distance from center of left tire to center of right tire, in cm (float)
  - ratio: ratio of sensor rotations to wheel rotations (e.g. 1 means 1:1, 3 means 3:1, etc.)
  - leftMotor: output port (A, B or C) to which left motor is attached
  - rightMotor: same as left
  - leftRot: input port (1, 2 or 3) to which left sensor is attached
  - rightRot: same as left

- Short form requires the first three above; assumes:
  - Motor.A and Motor.C
  - Sensor.S1 and Sensor.S3
Wheel diameters & distance calculation

• Use of the RotationNavigator class leaves the calculation of distance up to the framework

• Wheel diameters of the wheels found in the RCX kit include (all diameters in cm):
  – small yellow: 2.4  medium yellow: 3.0
  – large yellow: 4.3  small white: 3.04
  – medium white: 4.96  large white: 8.16
  – gray pulley: 3.0  tank tread: 2.8
Drive length & gear ratio

- Drive length: length of axle from wheel to wheel
- Gear ratio: number of times rotation sensor axle turns per full rotation of wheel; as mentioned previously, if the rotation sensor is mounted on the wheel axle, the ratio is 1:1
Accuracy & RotationNavigator

- Better than TimingNavigator, but not perfect
- Systematic Errors: most are reduced or eliminated; relative strengths of motors, battery life no longer issues
- Nonsystematic errors: terrain still a concern