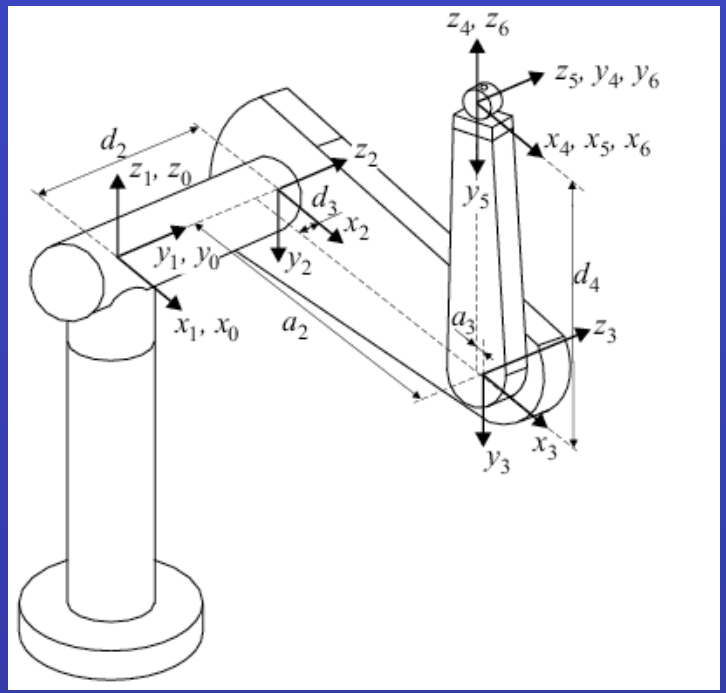
Research Week 2
Open Loop Contol
& Odometry

McGill

Odo-1

- Kinematic model
  - Relates the configuration of a robot in a local coordinate frame, e.g., (x,y) to motor shaft positions, e.g., (Ω_1,Ω_2).
  - A kinematic model is robot-specific.



- A robot with N degrees of freedom needs at least N motors/actuators
- The kinematic model describes the static behavior of a robot and does not include the effects of internal and external forces.
- Have to take into account the dynamic behavior as well.

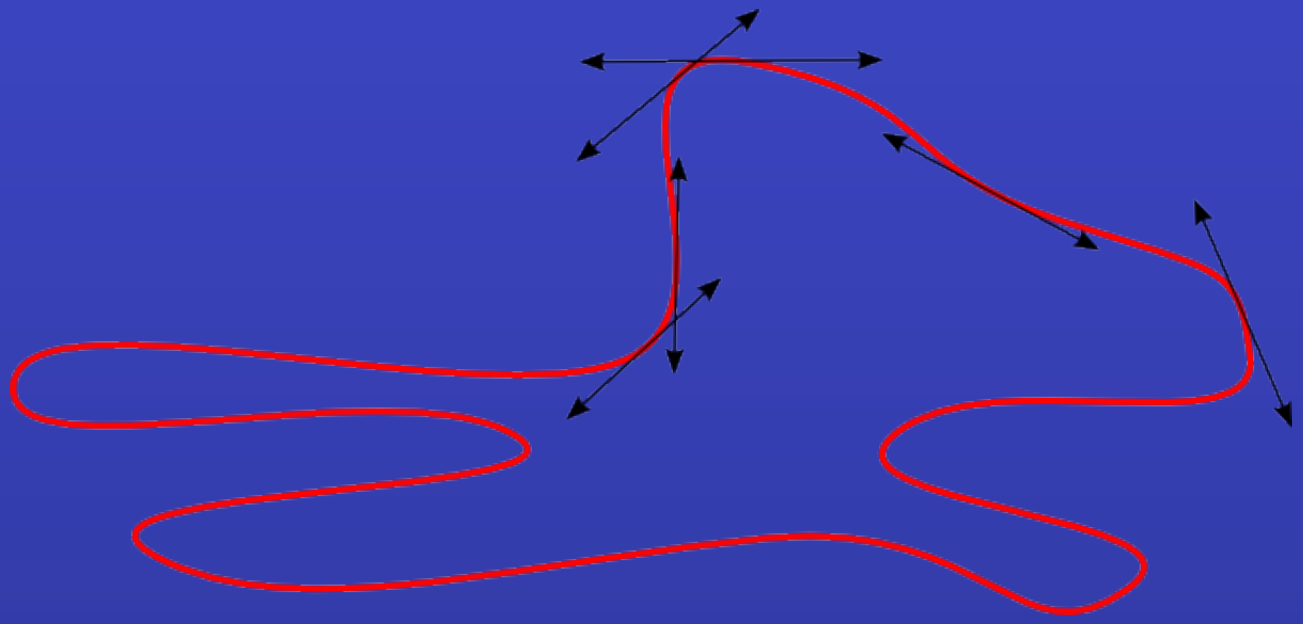
Odo-2

- Odometry
  - Determining the position of a robot in an external coordinate system based on counting motor rotations (change in actuator position).
  - The kinematic model is use to determine the instantanous displacement.
  - The integral of the displacement determines current position.
  - Odometry (sometimes referred to as dead reconing) provides only an *estimate* of the robot's actual position.
  - Odometry errors can be corrected by the use of sensors which measure the robot's true position with respect to landmarks in the environment (think of GPS).
  - In general, determining position involves simultaneous estimation over different sources of information, e.g., odometry, GPS, radar, etc.)

McGill

Odo-3

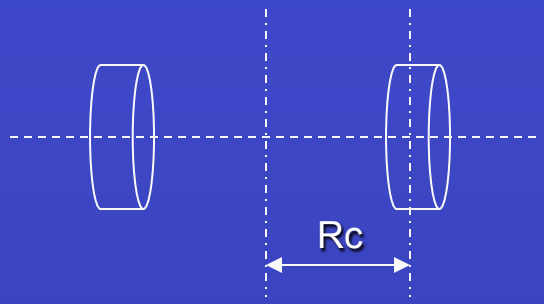
- Open loop control - using odometry to generate a path in space
- Method
  - Decompose path into tangent segments
  - Incrementally move along each segment (trajectory generator)

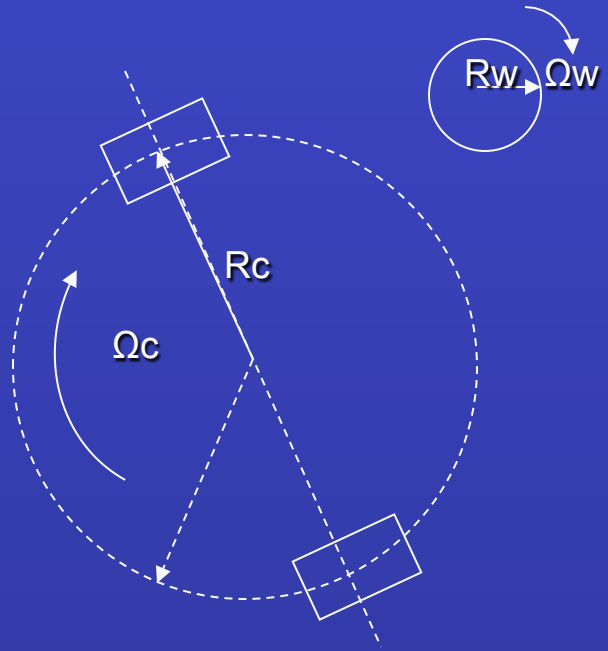


McGill

Odo-4

Odometry of the Tribot





Linear Motion:

- both wheels rotating
- distance = 2 π x wheel radius x Ωw / 360
- accuracy depends on degree of slip

Rotation in place (no displacement):

- Ωw defined as wheel rotation
- Ωc defined as rotation about cart center point
- Rw and Rc are radii of wheel and turning circle respectively (see diagrams at left)

arc length = radius x angle (radians)

Rw x Ωw = Rc x Ωc

Ωw = Ωc x (Rc / Rw)

Odo-5

Open Loop Control - Odometry, cont. 1

- For the Tribot, Rw = 2.8 cm, Rc = 6.61 cm, Rc/Rw = 2.35
- Putting this all together

  distance = 2 π Rw x Ωw / 360 - assuming wheel rotation Ωw in degrees

  - units for distance are the units of Rw

  Ωw = (360 D)/(2π Rw) = 20.46 D where D is in cm
- Practical problem – what if your computer does not support real numbers?

  Example: Assume we want to move by 5 cm

  Then Ωw = 20 x 5 = 100 (should be 102.3). This adds up over a long path.
- Integer scaling

  Multiply 20.46 by 100 to keep all 4 digits so scale factor is 2046

  Multiply 2046 x 5 = 10230, and then divide by 100, 10230 / 100 = 102

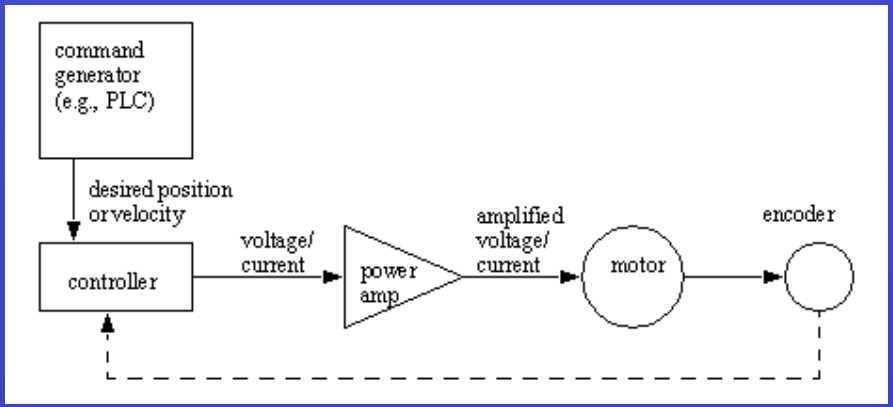
  Observe that the answer is equivalent to determining the exact answer and then converting to integer.
- General rule

  Scale to maintain full precision, perform computation, then scale result back.

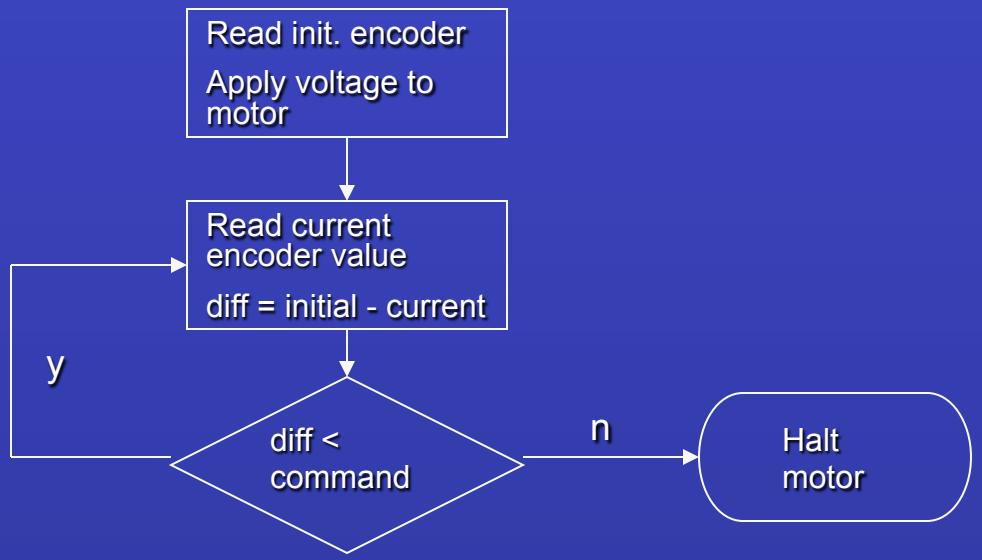
McGill

Odo-6

Servo Motors



Basic Ideas



Odo-7

Methods for dealing with servos

```
rotate(int angle, boolean return)  // servos relative to current
                                   // angular position; angle is
                                   // degrees.  Optional return
                                   // arg sets blocking on/off.

rotateTo(int angle, boolean return) // angle is absolute here

setSpeed(int speed)                // set speed, degrees/sec

regulateSpeed(boolean yes)         // default is to regulate

smoothAcceleration(boolean yes)    // enables smoother acceleration
```

Threading

Motor servo methods run in independent threads and can (usually) be set to block or run asynchronously.

Methods available for synchronizing motors using the synchronizeWith method.

Odo-8

A Simple Open Loop Demonstration

```
import lejos.hardware.ev3.LocalEV3;
import lejos.hardware.lcd.TextLCD;
import lejos.hardware.motor.EV3LargeRegulatedMotor;
import lejos.hardware.port.MotorPort;

// Simple trajectory generation using a two-wheeled cart

public class OpenLoop {

// class variables

static final int DISTTODEG=21;         // 360/(2xPixRw)  Rw=2.8cm       (20)
static final int ORIENTTODEG=305;  //(Rc/Rw)*100     Rc=7.94cm   (283)
static final int SIDE=5000;              // Dimension of square in units of 0.1cm
static final int FWDSPEED=180;       // Forward speed of robot (deg/sec)
static final int TRNSPEED=90;         // Rotational speed of robot (deg/sec +/-)

// static objects (resources that are shared)

static TextLCD t = LocalEV3.get().getTextLCD();              // LCD
static EV3LargeRegulatedMotor leftMotor = new EV3LargeRegulatedMotor(MotorPort.A);
static EV3LargeRegulatedMotor rightMotor = new EV3LargeRegulatedMotor(MotorPort.D);
```

Odo-9

A Simple Open Loop Demonstration cont.

```
 public static void main(String[] args) {
        t.clear();                                  // Clear display
        t.drawString("Open Loop Test", 0, 0);   // Print banner
        OpenLoop demo = new OpenLoop();
        demo.go();                                  // And do it
}

public void go() {
        MoveDistFwd(SIDE,FWDSPEED);         // Draw a side
        Rotate(90,TRNSPEED);                //Rotate90°(repeat4times)
        MoveDistFwd(SIDE,FWDSPEED);
        Rotate(90,TRNSPEED);
        MoveDistFwd(SIDE,FWDSPEED);
        Rotate(90,TRNSPEED);
        MoveDistFwd(SIDE,FWDSPEED);
        Rotate(90,TRNSPEED);
}
```

Odo-10

A Simple Open Loop Demonstration cont.

```
public void MoveDistFwd(int distance, int speed) {
        int WRotationAngle;

// Pay attention to blocking vs. non-blocking motor methods.
// To get 2 motors to start at the same time, first call must
// be non-blocking.

        WRotationAngle=distance*DISTTODEG/100;     // Dist to turns
        leftMotor.setSpeed(speed);                 // Forward L+R
        rightMotor.setSpeed(speed);
        leftMotor.rotate(WRotationAngle,true);     // Apply turns
        rightMotor.rotate(WRotationAngle);
}
public void Rotate(int angle, int speed) {
        int CRotationAngle;

        CRotationAngle=angle*ORIENTTODEG/100;      // Orient to turns
        leftMotor.setSpeed(speed);                 // Slower in place
        rightMotor.setSpeed(speed);
        leftMotor.rotate(CRotationAngle,true);     // Left, non block
        rightMotor.rotate(-CRotationAngle);;       // Right can
}}
```

McGill

Odo - 11

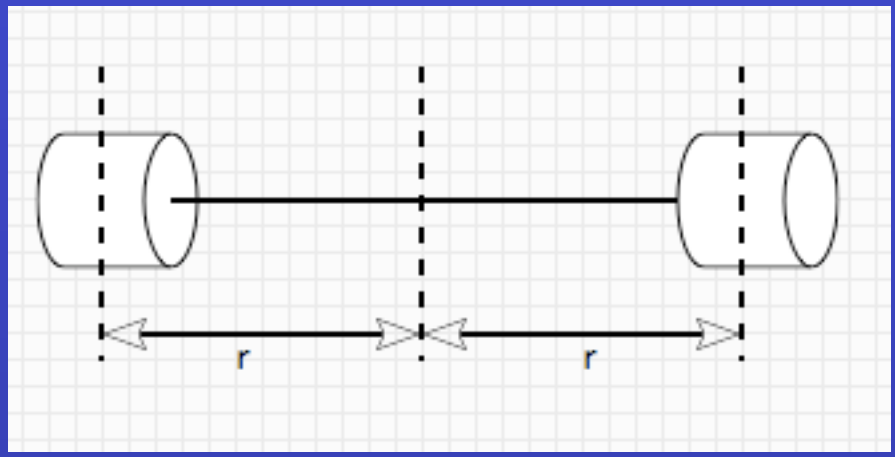
Observations

- The accuracy of the path depends on many factors
  - The amount of wheel slip against the floor
  - Motor synchronization (or lack thereof)
  - Control over acceleration and deceleration (sudden starts and stops throw the vehicle off it's trajectory.
  - Experiments with the Tribot in running a closed loop (square) show that return to the origin can be off from anywhere from 5 to 100 cm depending on the flooring.

McGill

Odo - 12

Tracking Position via Odometry



Inputs

- Rotations of left and right wheels
- Radii of wheels
- Wheelbase of vehicle

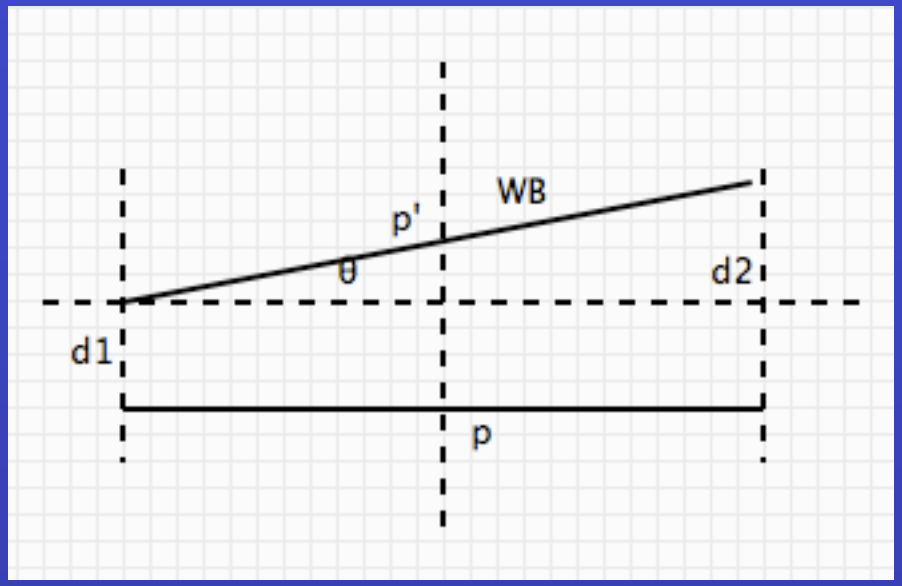
Outputs (a vector)

- Magnitude (length) of displacement
- Change in heading
- Compute new position from above

McGill

Odo - 13

Calculating instantaneous displacement from measurements



Calculations:

- Measure Φl, Φr
- Calculate d1 = (Rw π Φl)/180
- Calculate d2 = (Rw π Φlr/180
- Calculate d = d2 - d1
- Calculate θ ≈ d / WB

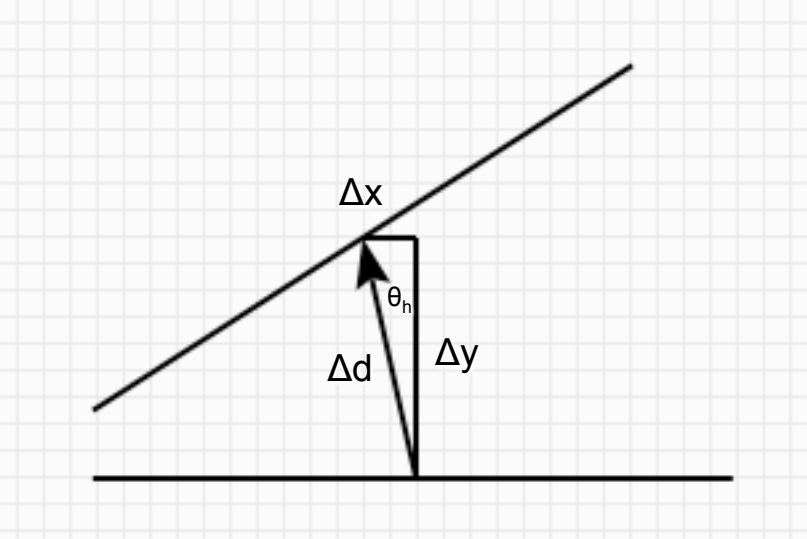
n.b. θ is in *radians*

- New heading: θ_h = old heading + θ
- Displacement (magnitude): $d_h \approx (d1 + d2) / 2$

McGill

Odo - 14

Updating position



Calculations:

- $\Delta x = \Delta d \sin \theta_h$
- $\Delta y = \Delta d \cos \theta_h$

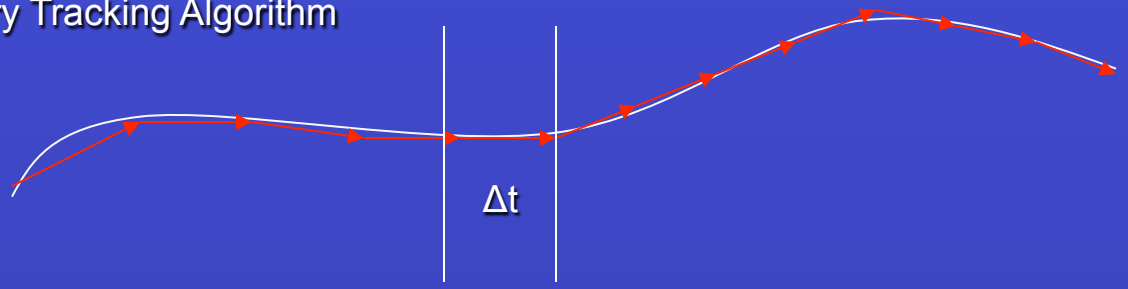
Current Position:

- $x_{current} = x_{last} + \Delta x$
- $y_{current} = y_{last} + \Delta y$

McGill

Odo - 15

Trajectory Tracking Algorithm



- Initialize
  - Set position to starting point (default = 0,0)
  - Read left and right tacho counts. Save as last_tacho_l and last_tacho_r respectively.
- Updating
  - Read left and right tacho counts. Determine Φ_l and Φ_r. Update last_tacho_l and r.
  - Calculate d1, d2, d, and $\theta \Rightarrow \theta_h$, d_h.
  - Calculate Δx, Δy to determine current position.
  - Wait Δt
  - Back to Updating step

n.b. Instructions are assumed to execute on the order of microseconds, i.e., negligible relative to the Wait instruction.

Odo - 16

```
import lejos.hardware.Button;
import lejos.hardware.ev3.LocalEV3;
import lejos.hardware.lcd.TextLCD;
import lejos.hardware.motor.EV3LargeRegulatedMotor;
import lejos.hardware.port.MotorPort;
import lejos.utility.Timer;
import lejos.utility.TimerListener;

//
// A simple odometry demonstration using the EV3 controller
//

public class OdoDemo implements TimerListener {

// Class Constants

public static final int SINTERVAL=50;          // Period of sampling f (mSec)
public static final int SLEEPINT=500;          // Period of display update (mSec)
public static final double WB=16.0;            // Wheelbase (cm)
public static final double WR=2.7;             // Wheel radius (cm)
```

Odo - 17

```
// Class Variables

public static int lastTachoL;        // Tacho L at last sample
public static int lastTachoR;        // Tacho R at last sample
public static int nowTachoL;         // Current tacho L
public static int nowTachoR;         // Current tacho R
public static double X;              // Current X position
public static double Y;              // Current Y position
public static double Theta;          // Current orientation

// Resources

static TextLCD t = LocalEV3.get().getTextLCD();
static EV3LargeRegulatedMotor leftMotor = new EV3LargeRegulatedMotor(MotorPort.A);  // L
static EV3LargeRegulatedMotor rightMotor = new EV3LargeRegulatedMotor(MotorPort.D); // R
```


```
public static void main(String[] args) throws InterruptedException {

        boolean noexit;
        int status;

// Set up display area

        t.clear();
        t.drawString("Odometer Demo",0,0,false);
        t.drawString("Current X  ",0,4,false);
        t.drawString("Current Y  ",0,5,false);
        t.drawString("Current T  ",0,6,false);

// Set up timer interrupt.  Creates a thread which services the timer interrupt.  Here
// control will be transfered to OdoDemo.timedOut().

        Timer myTimer = new Timer(SINTERVAL,new OdoDemo());

// Create thread to move the cart simultaneously with odometer operation
// (You will need to create a doSquare class for this that extends thread)

        doSquare cart = new doSquare(leftMotor, rightMotor);
```

Odo - 19

```
// Clear tacho counts and put motors in freewheel mode.  Then initialize tacho count
// variable to its current state.
                leftMotor.resetTachoCount();
                rightMotor.resetTachoCount();
                lastTachoL=leftMotor.getTachoCount();
                lastTachoR=rightMotor.getTachoCount();

// Enable timer interrupts (i.e. start the odometer), and start the cart moving.
                myTimer.start();                                    //  These are both threads
                cart.start();                                       //  that operate concurrently

// Enter display loop.  Terminate on any button push.  (This is actually thread #3)
                noexit=true;
                while(noexit) {
                        status=Button.readButtons();        // Terminate on ENTER button
                        if (status==Button.ID_ENTER) {
                                System.exit(0);
                        }
                        t.drawInt((int)X,4,11,4);                   // Current X estimate
                        t.drawInt((int)Y,4,11,5);                   // Current Y estimate
                        t.drawInt((int)(Theta*57.2598),4,11,6);   // Current heading
                        Thread.sleep(SLEEPINT);                     // Put thread to sleep
                }
        }
```

Odo - 20

```
// The "odometer" is implemented in the timer listener (aka interrupt service routine).
// It follows the recipe described in the class notes.

        public void timedOut() {
                double distL, distR, deltaD, deltaT, dX, dY;

                nowTachoL = leftMotor.getTachoCount();        // get tacho counts
                nowTachoR = rightMotor.getTachoCount();
                distL = 3.14159*WR*(nowTachoL-lastTachoL)/180;     // compute wheel
                distR = 3.14159*WR*(nowTachoR-lastTachoR)/180;   // displacements
                lastTachoL=nowTachoL;          // save tacho counts for next iteration
                lastTachoR=nowTachoR;
                deltaD = 0.5*(distL+distR);       // compute vehicle displacement
                deltaT = (distL-distR)/WB;        // compute change in heading
                Theta += deltaT;                  // update heading
                dX = deltaD * Math.sin(Theta);    // compute X component of displacement
                dY = deltaD * Math.cos(Theta);  // compute Y component of displacement
                X = X + dX;                       // update estimates of X and Y position
                Y = Y + dY;
        }
}
```