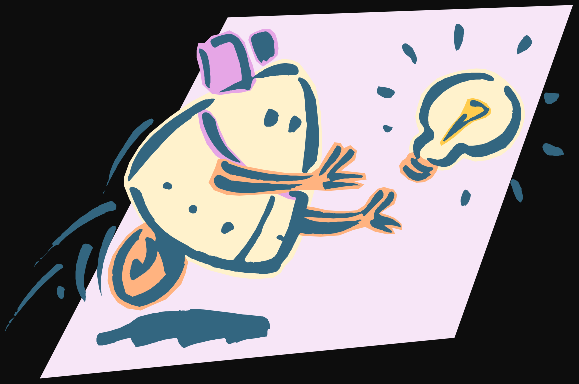
Homing and Tracking

Homing Behavior

- ***Homing*** involves orienting or directing homeward to a destination
- ***Taxis***: A steering toward or away from some directional stimulus or a gradient of stimulus intensity. (E.g., seeking out light, temperature, energy etc..)
- Used to orient the robot towards or away from something progressively.
- There are three main types:
  1. Klinotaxis
  2. Tropotaxis
  3. Menotaxis

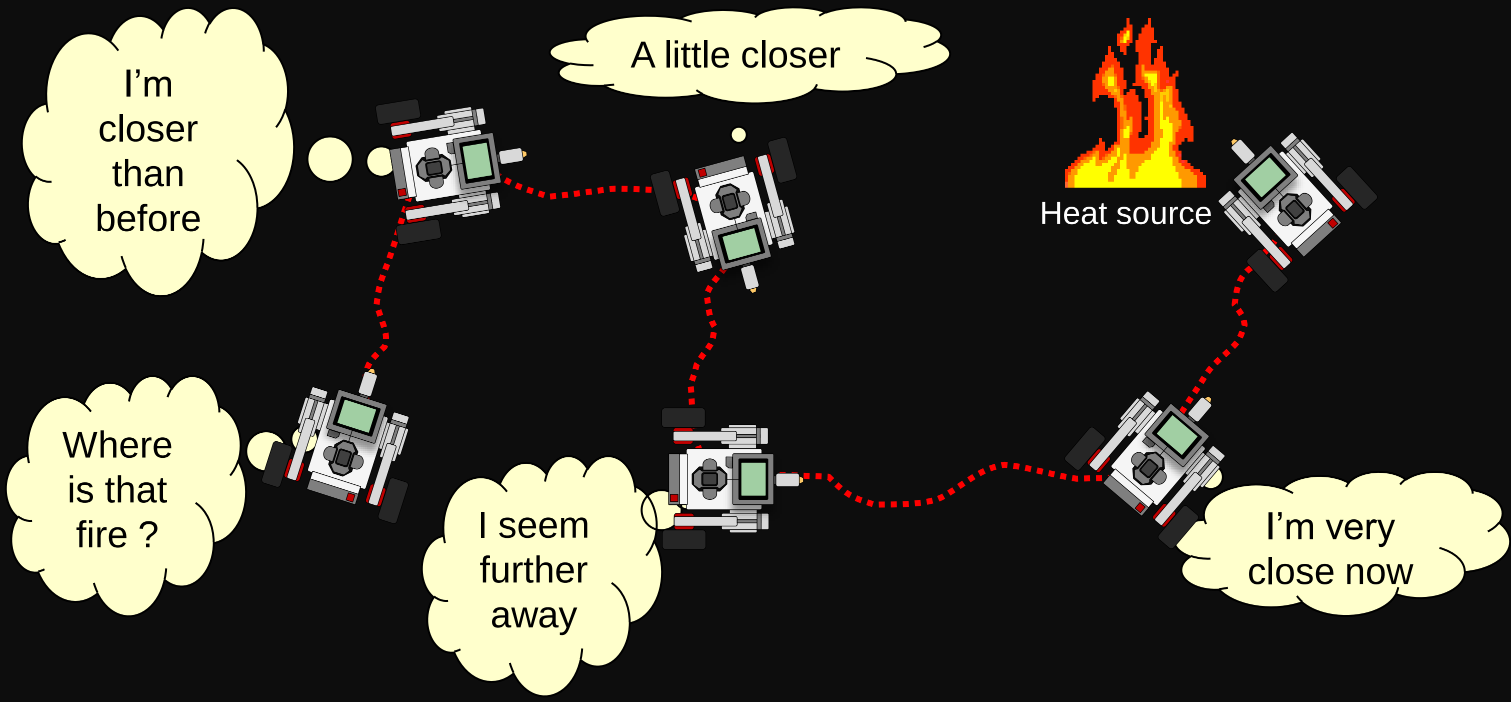


Homing Behavior

1. Klinotaxis:

Taking sensor readings at various locations in sequence in order to head towards a stimulus

e.g., Temperature sensing or “sniffing” out chemicals

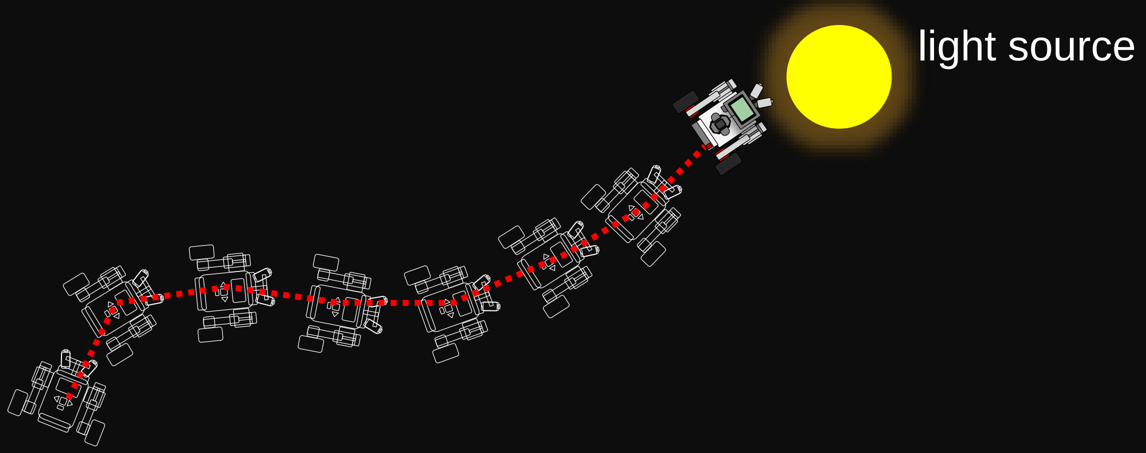


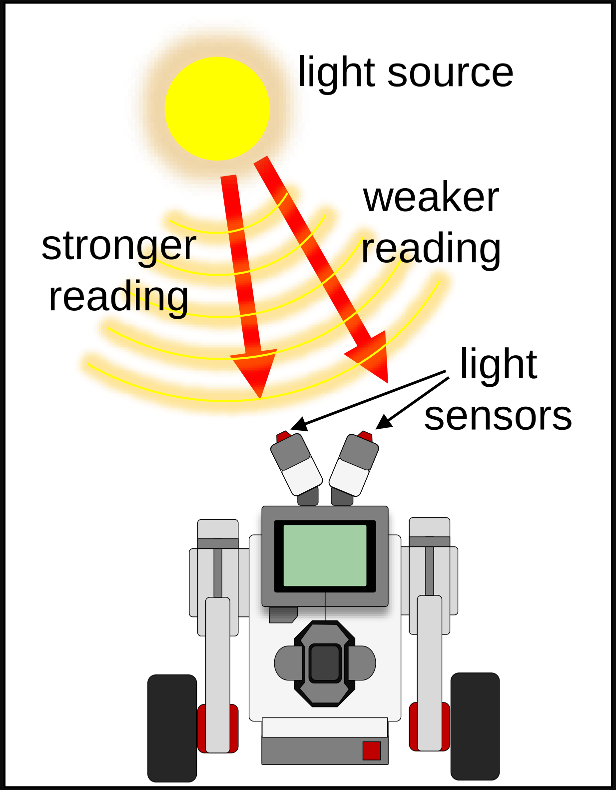
Homing Behavior

2. Tropotaxis:

Using the difference between two similar sensors to determine the direction of a certain stimulus

e.g., seeking out a light source



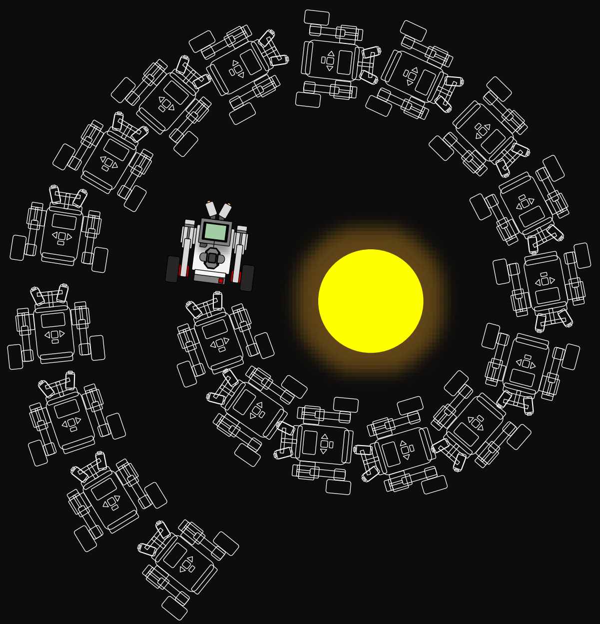


Homing Behavior

3. Menotaxis:

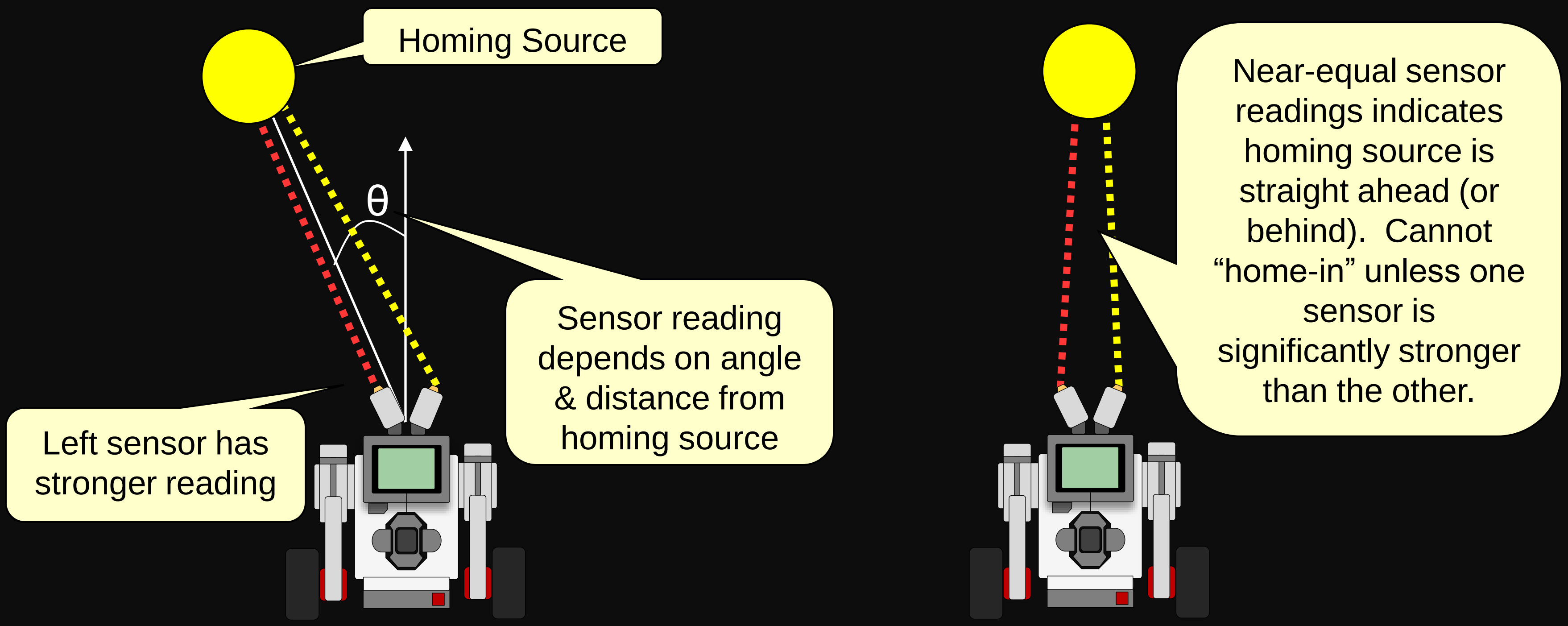
Maintaining a fixed angle between the path of motion and the direction of the sensed stimulus

e.g., spiraling around a light source



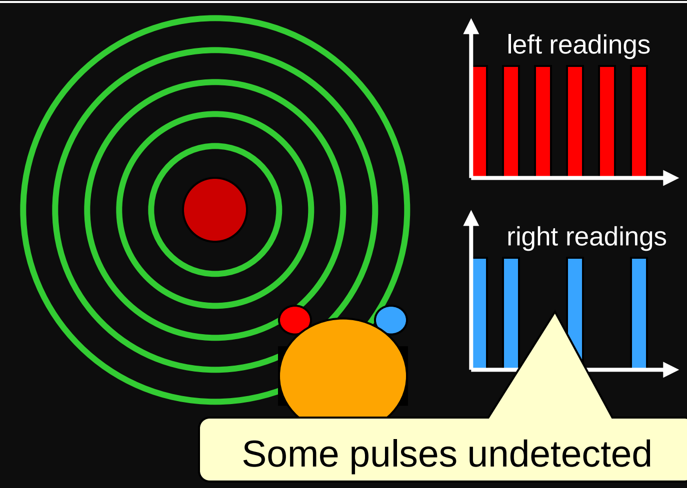
Tropotaxis Homing Logic

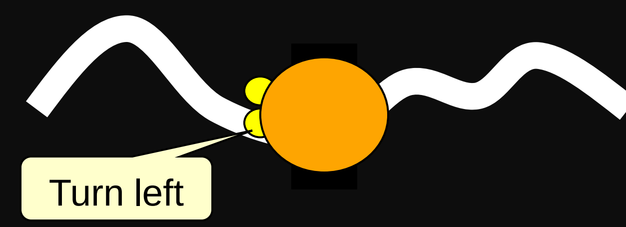
- Easiest with 2 sensors whose readings increase when pointed towards homing source (tropotaxis):

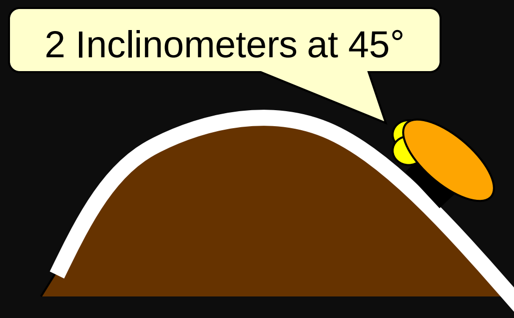


Other Types of Homing

- Other forms of homing-in:
  - Beacon Following
    - Beacon emits pulsed signal which is more reliably detected by closer sensor
  - Line Following
    - Photodetectors read stronger on white, can detect when a sensor leaves black line
  - Hill Climbing
    - Climb hill by minimizing roll while keeping pitch positive using inclinometers

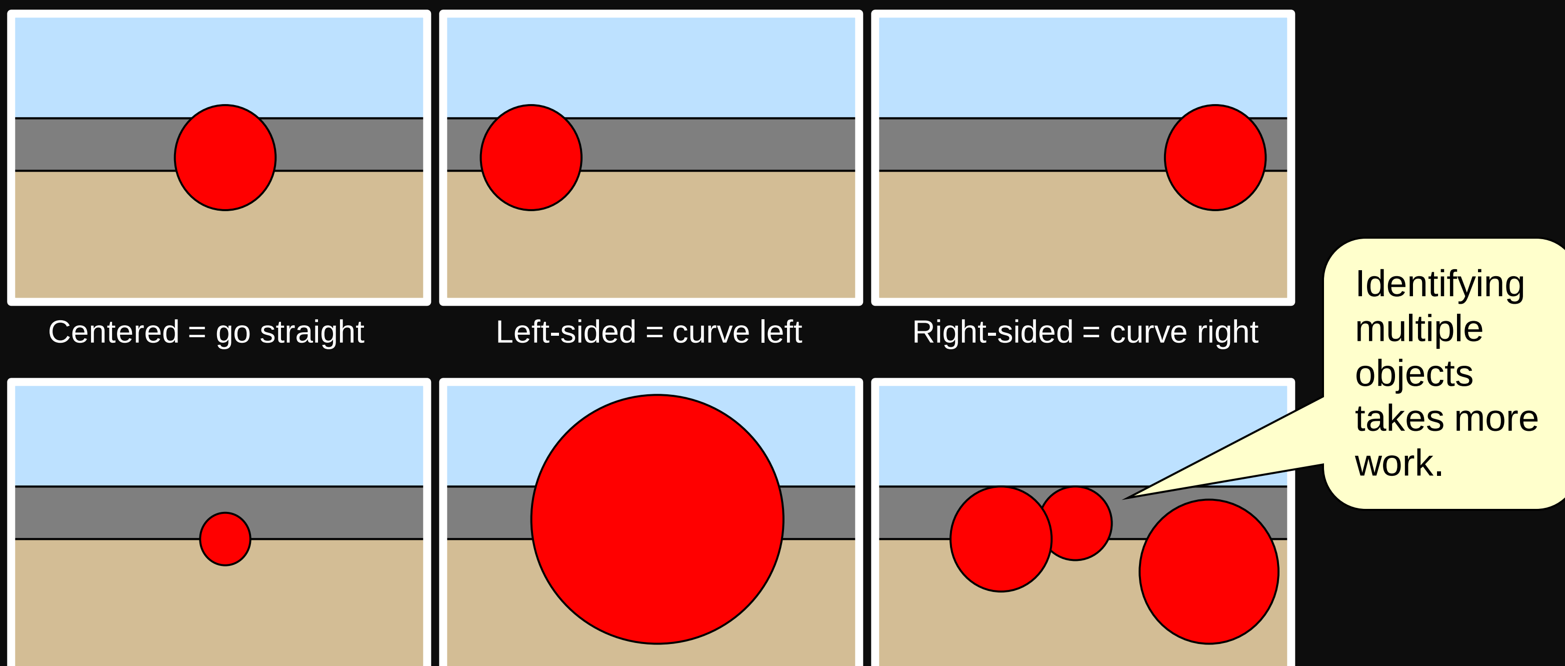






Camera Tracking

- Cameras are often used to track objects and can be used to find, locate and "home-in" on them.
- Can look for colored "blobs" and make decisions based on their location.



Centered = go straight

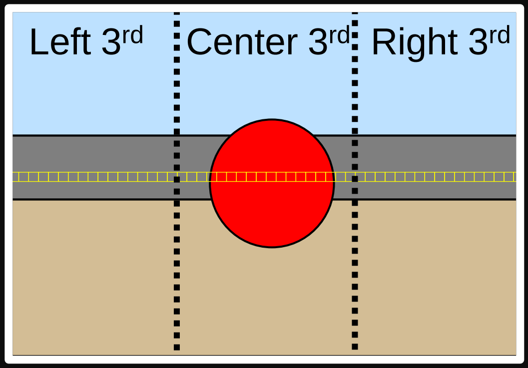
Left-sided = curve left

Right-sided = curve right

Less red = further

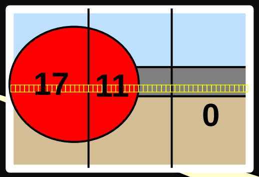
More red = closer

Basic Tracking: Checking Pixels



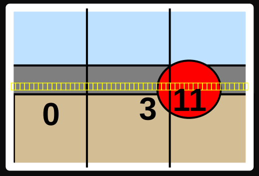
- Grab a single row of pixels in the center of the image and count the **amount of red** in each of 3 zones:

```
If (leftCount > (rightCount + ε))
      then the object is on
      the left side
```

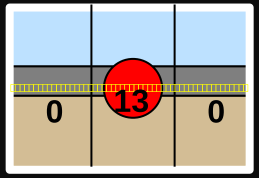


Allow for a small epsilon **ε** so that robot doesn't zig-zag back and forth due to minor noise fluctuations.

```
If (rightCount > (leftCount + ε))
      then the object is on
      the right side
```

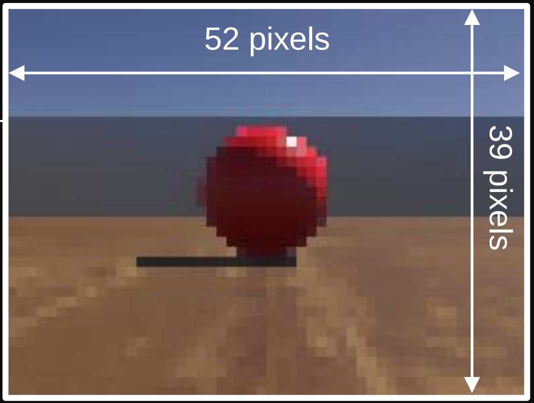



```
If (centerCount > ε)
     then there is red ahead
```



- If none of above are true, then no object is detected

E-puck Camera



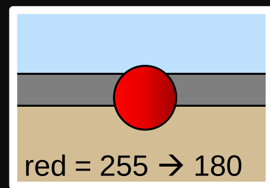
- The e-puck robot has a color camera that can be used for processing images in a simple manner:

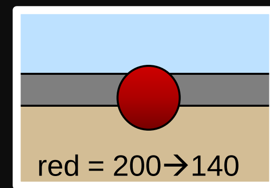
```
import com.cyberbotics.webots.controller.Camera;

// Cameras are objects
Camera  camera;

// Set up the camera
camera = = new Camera("camera");
camera.enable(timeStep);

// WHILE LOOP {

       // Need to capture an image … comes back as
       // a 1D array with rows one after the other
       int[] image = camera.getImage();

       // Can get the red, green and blue value of a pixel at position (x, y) in the array
       // where (0,0) is at the top left of the image
       int r = Camera.imageGetRed(image, 52, x, y);  // 52 is image width
       int g = Camera.imageGetGreen(image, 52, x, y);
       int b = Camera.imageGetBlue(image, 52, x, y);
// }
```

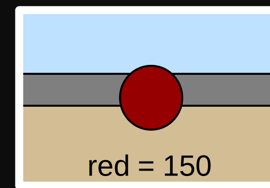
This code MUST be in the main **while** loop, otherwise a runtime exception will occur.

Camera Colors

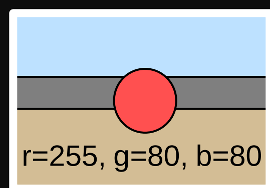
- Objects will have shadows… will affect the red color value:

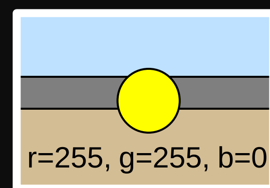


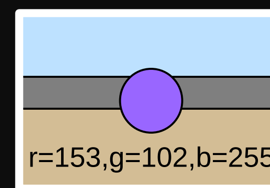




- Also, other colors have red in them:







- So, need to check all 3 color components to decide if red:

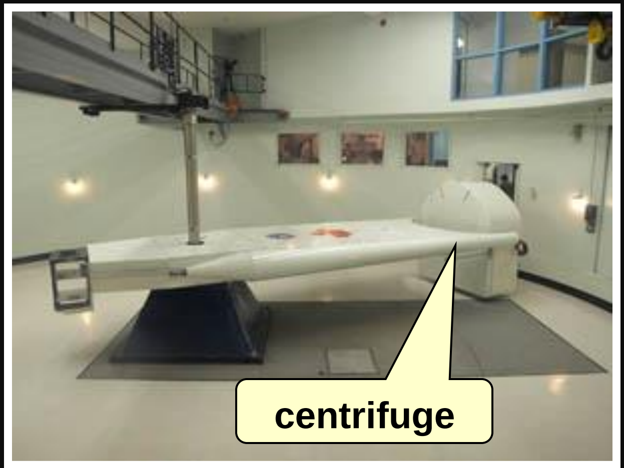
```
if ((red > 60) && (green < 100) && (blue < 100)) ...
```

Acceleration

- Acceleration is the rate of change in velocity
  - Measured in (meters per second) per second. (i.e., m/s^2)
- A "g" is a unit of acceleration equal to the earth's gravity at sea level:

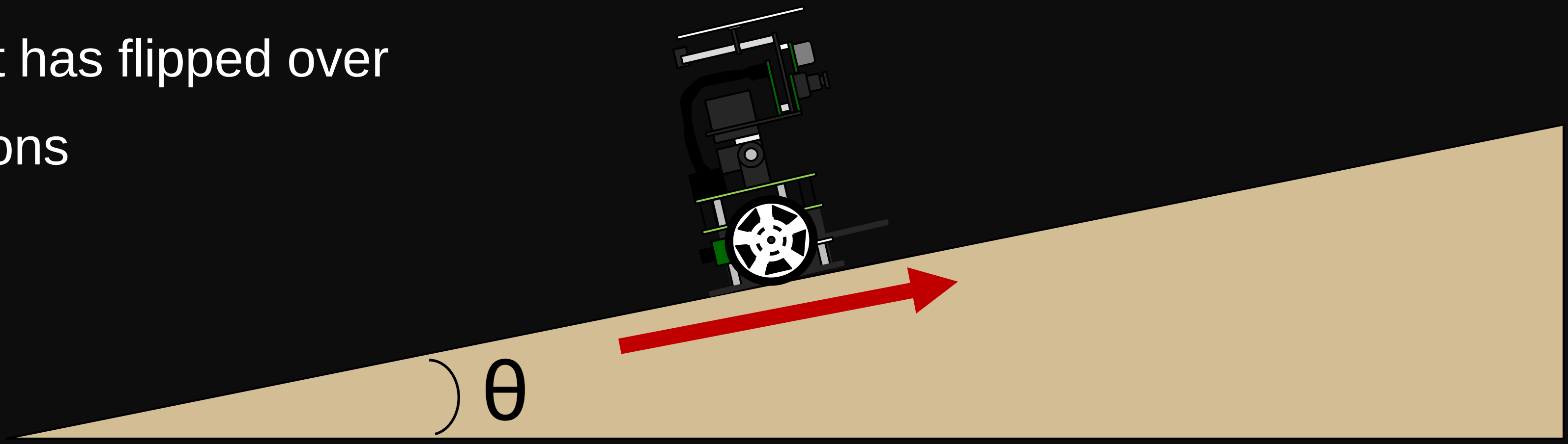
$$g = 9.81\ m/s^2 \quad (\text{or } 32.2\ ft/s^2)$$

1g - Earth's gravity
2g - Passenger car when taking a corner
2g - Bumps in the road
3g - Indy car driver when taking a corner
5g - Bobsled rider when taking a corner
7g - Unconsciousness
10g - Space shuttle



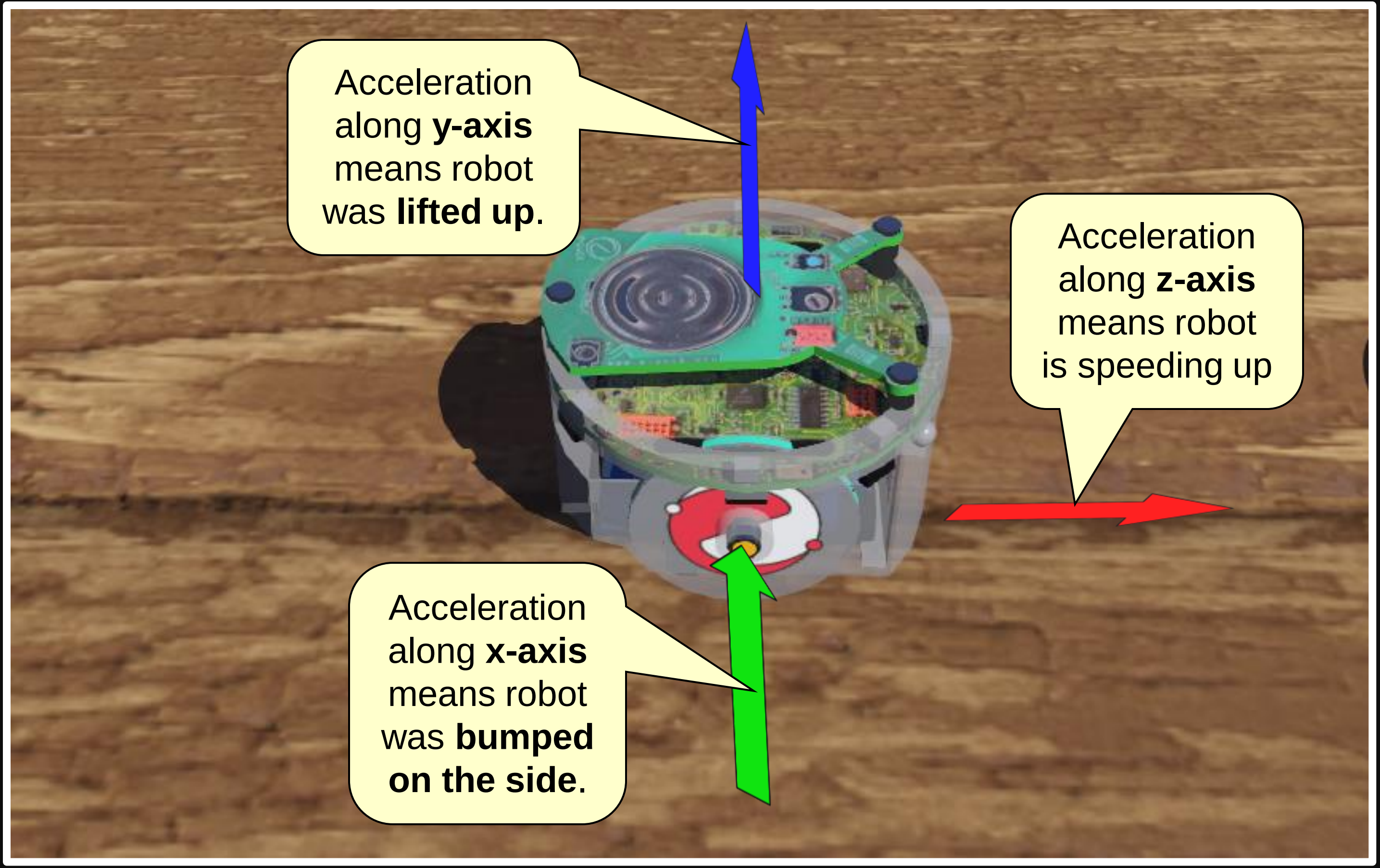
Accelerometers

- An accelerometer can measure:
  - Static acceleration forces
    - (e.g., force of gravity)
  - Dynamic acceleration forces
    - (e.g., vibrations of the device)
- Can determine:
  - angle of incline
  - if robot has flipped over
  - vibrations



E-Puck Acceleration Detection

- As sensor moves, it detects acceleration (i.e., change in speed) in one of the three axis directions.



E-Puck Accelerometer Angles

(0.0, 0.0, 9.8) | (0.0, 0.0, 9.8) | (0.0, 0.0, 9.8) | (0.0, 0.0, 9.8)

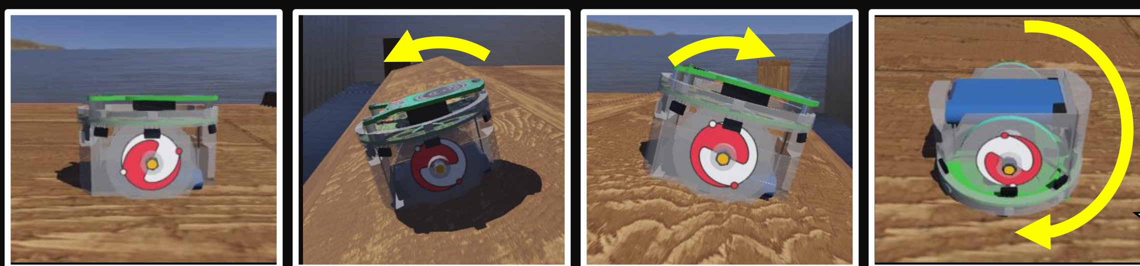
Spin about Y-axis

(0.0, 0.0, 9.8)

Tip Forward (**-2.6**, -0.0, 9.5)

Tip Backward (**2.6**, -0.0, 9.5)

Upside Down (0.0, -0.0, **-9.8**)



Spin about X-axis

(0.0, 0.0, 9.8)

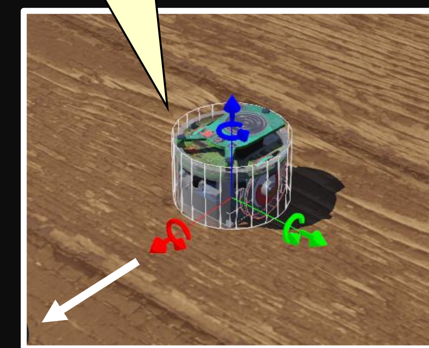
Tip Left (-0.0, **-2.5**, 9.5)

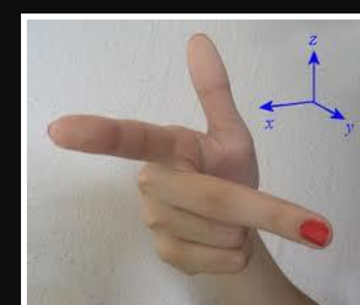
Tip Right (0.0, **2.5**, 9.5)

Upside Down (0.0, -0.0, **-9.8**)

Spin about Z-axis

3-Axis accelerometer





E-Puck Accelerometer Code

```
import com.cyberbotics.webots.controller.Accelerometer;

// Accelerometers are objects
Accelerometer  accelerometer;

// Set up the accelerometer
Accelerometer accelerometer = new Accelerometer("accelerometer");
accelerometer.enable(timeStep);

// Need to capture (x, y, z) values in a double array
double[] accelValues = new double[3];

// WHILE LOOP {
    // Get all three values each time
    accelValues = accelerometer.getValues();
    String s = String.format("Accel (x=%2.1f, y=%2.1f, z=%2.1f) ",
                          accelValues[0],accelValues[1],accelValues[2]);
    System.out.println(s);
// }
```

```
Accel (x=-0.01, y=0.01, z=9.81)
Accel (x=-0.01, y=0.01, z=9.78)
Accel (x=-0.01, y=0.02, z=9.78)
Accel (x=-0.03, y=-0.61, z=9.82)
Accel (x=-0.09, y=0.02, z=9.79)
Accel (x=-0.09, y=0.02, z=9.79)
Accel (x=-0.02, y=-2.46, z=9.75)
Accel (x=0.20, y=1.83, z=9.82)
Accel (x=-0.91, y=-1.98, z=6.91)
Accel (x=-1.60, y=-1.31, z=14.02)
Accel (x=0.86, y=-0.58, z=7.40)
Accel (x=-0.32, y=-0.41, z=11.56)
Accel (x=0.28, y=3.32, z=8.96)
Accel (x=-0.20, y=-3.23, z=9.80)
Accel (x=-0.03, y=0.37, z=10.01)
Accel (x=-0.55, y=-1.59, z=9.29)
Accel (x=0.03, y=0.66, z=6.68)
Accel (x=0.16, y=-6.35, z=13.63)
Accel (x=-1.48, y=-2.26, z=5.56)
Accel (x=-1.00, y=-4.67, z=14.12)
Accel (x=-1.18, y=-0.75, z=7.32)
Accel (x=-1.50, y=-2.68, z=9.41)
Accel (x=-1.23, y=-0.35, z=9.38)
Accel (x=-1.36, y=-1.87, z=9.40)
```

As robot moves, values will fluctuate a lot.

Accelerometer Data Smoothing

- With data bouncing up and down too much, we need to smooth it out by taking a ***running average***:
  - Initialize an array of size 10 or so:

0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9

```
double[] readings;
readings = new double[10];
```

  - As readings come in, fill up the array:

0.01	0.01	0.02	-0.61	0.02	0.02	-2.46	1.83	-1.98	-1.31
0	1	2	3	4	5	6	7	8	9

```
readings[i] = accelValues[1];
```

  - When 11th reading comes in, wrap around to the start again, overwriting the oldest readings:

-0.58	0.01	0.02	-0.61	0.02	0.02	-2.46	1.83	-1.98	-1.31
0	1	2	3	4	5	6	7	8	9

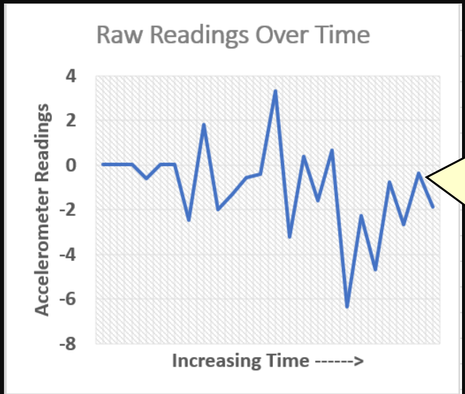
```
i = (i + 1) % 10;
```

Accelerometer Data Smoothing

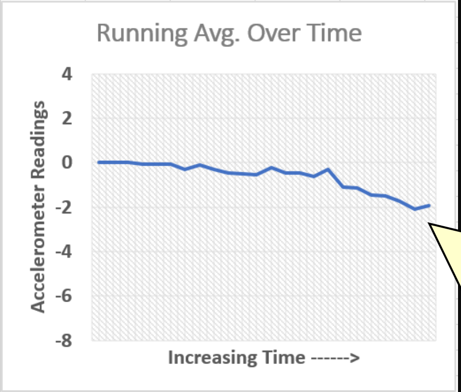
- When we take the average of the array, we get the average of the latest 10 readings:

This is the running average.

0	1	2	3	4	5	6	7	8	9	Average
0.01	0.01	0.02	-0.61	0.02	0.02	-2.46	1.83	-1.98	-1.31	-0.445
-0.58	0.01	0.02	-0.61	0.02	0.02	-2.46	1.83	-1.98	-1.31	-0.504
-0.58	-0.41	0.02	-0.61	0.02	0.02	-2.46	1.83	-1.98	-1.31	-0.487
-0.58	-0.41	3.32	-0.61	0.02	0.02	-2.46	1.83	-1.98	-1.31	-0.114
-0.58	-0.41	3.32	-3.23	0.02	0.02	-2.46	1.83	-1.98	-1.31	-0.769
-0.58	-0.41	3.32	-3.23	0.37	0.02	-2.46	1.83	-1.98	-1.31	-0.409
-0.58	-0.41	3.32	-3.23	0.37	-1.59	-2.46	1.83	-1.98	-1.31	-0.605



It is hard to tell exactly what is going on due to the shaking of the robot.



Noise is eliminated. Since this is the y-axis, we can detect that the robot is starting to tip forward.

Start the Lab …