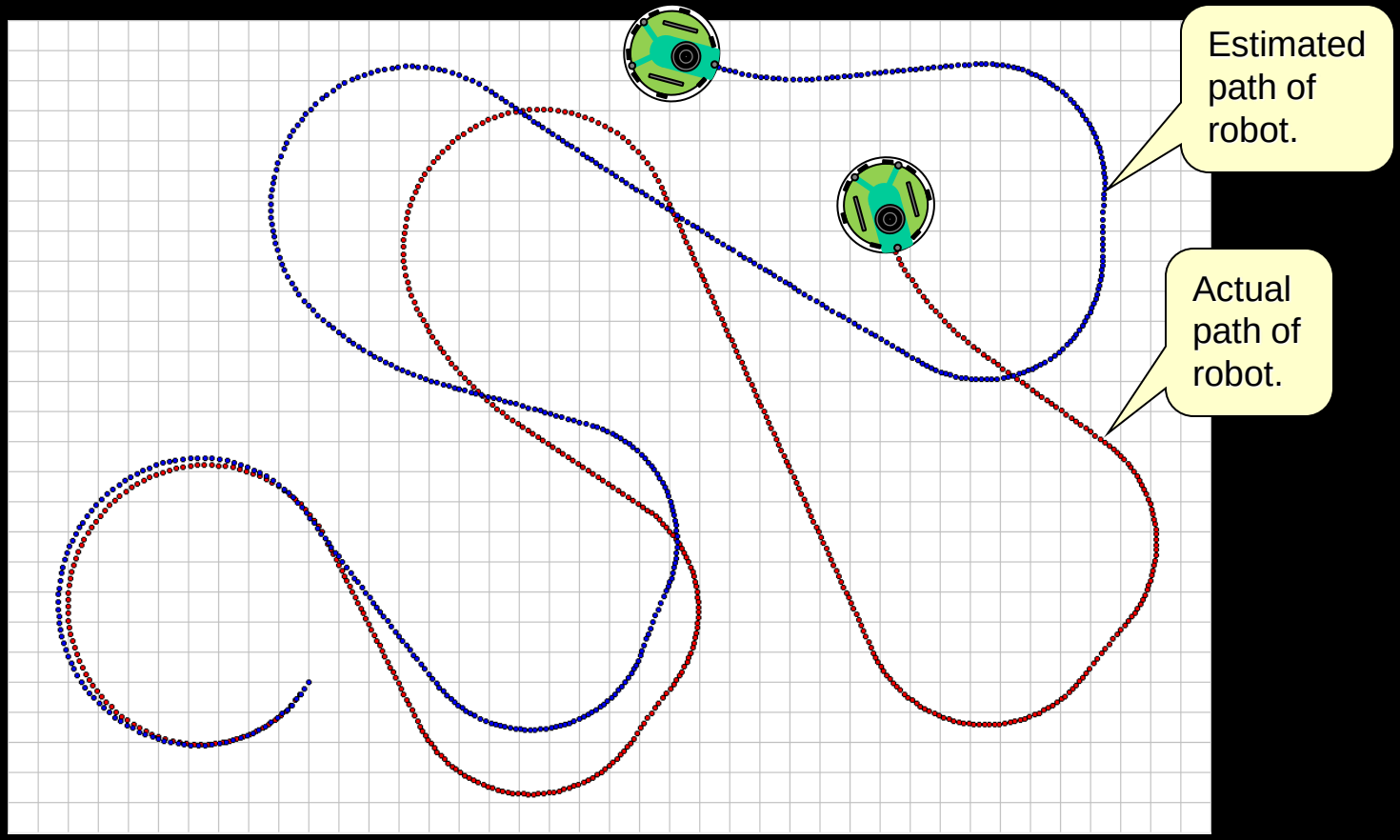
Odometry Correction

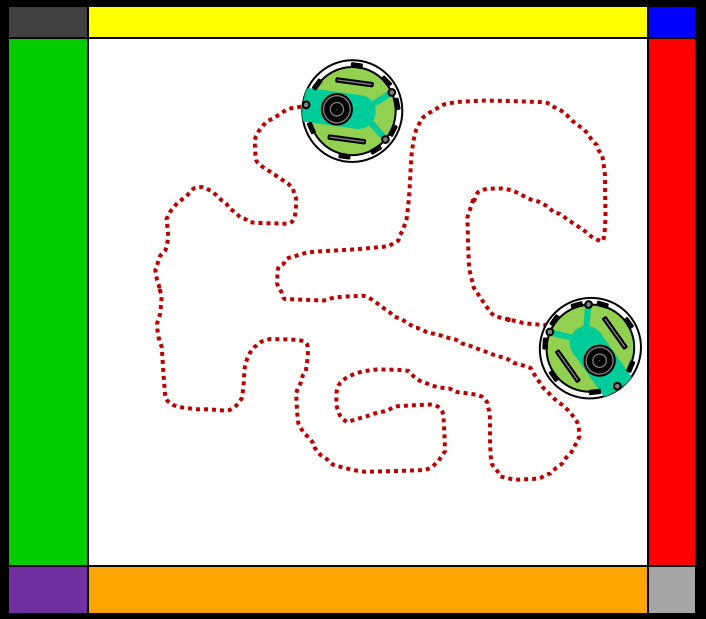
Odometry Problems

- Recall the odometry error … unpleasant indeed.



Grid-Based Estimation Problems

- Grid-based estimation helps (i.e., error is bounded)
- But it requires modification of environment (i.e., must install colored tiles)
- And, if we travel too long on cell boundaries, we lose our spot!
- Also, we cannot determine any position changes within a cell.
  - Can be serious if we are mapping or trying to accomplish some task within the cell.

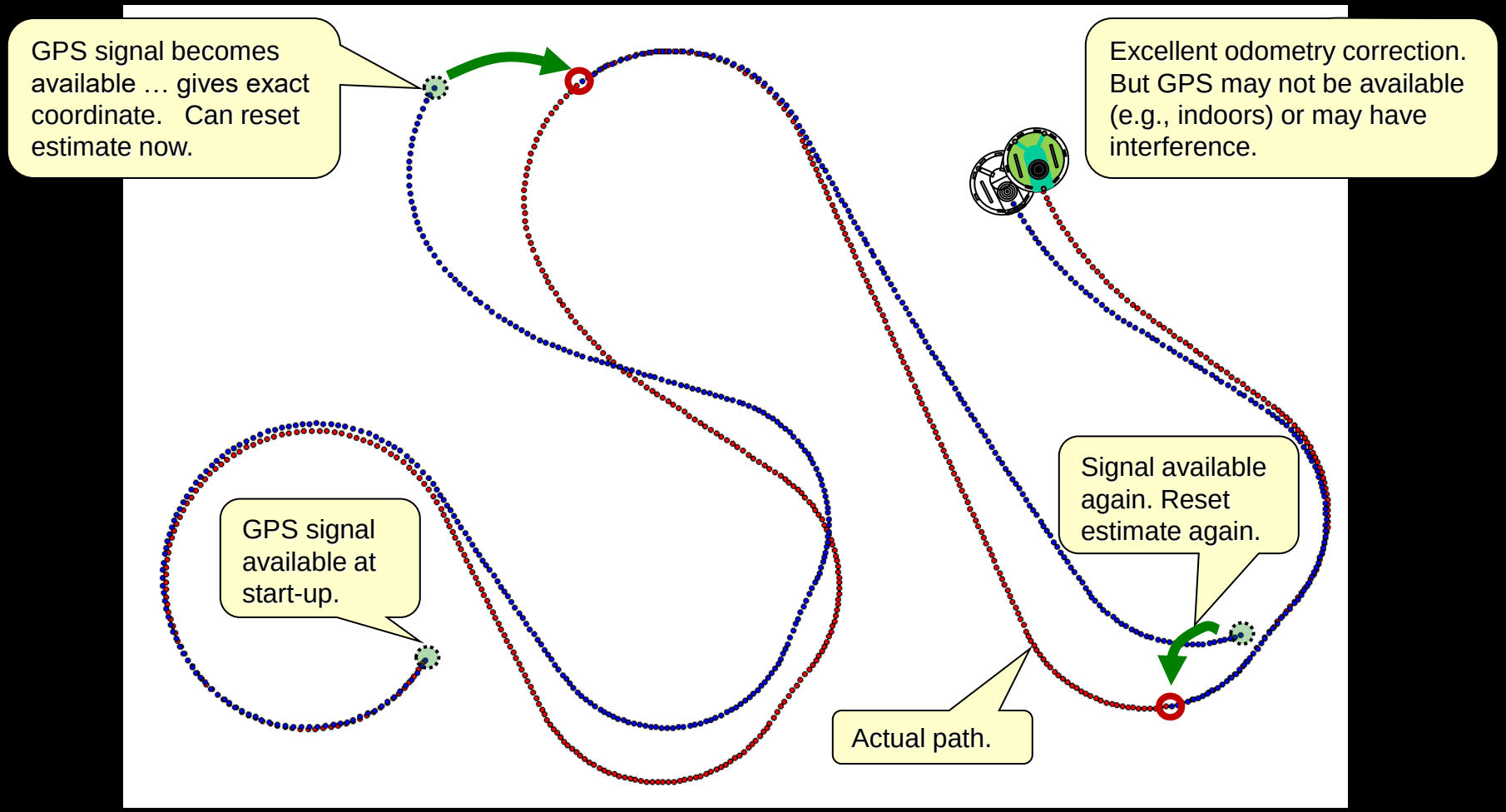


Odometry Correction

- Need to find a way to prevent error from growing too large over time by re-adjusting estimates when they become too far off.
- But how do we know when the estimate error has grown too much ?
  - Need to compare with some other known data (e.g., gps, compass reading, map, beacons, other estimates, etc…)
  - The more accurate this "other" data is … the more accurately we will be able to reset the robot's estimate.

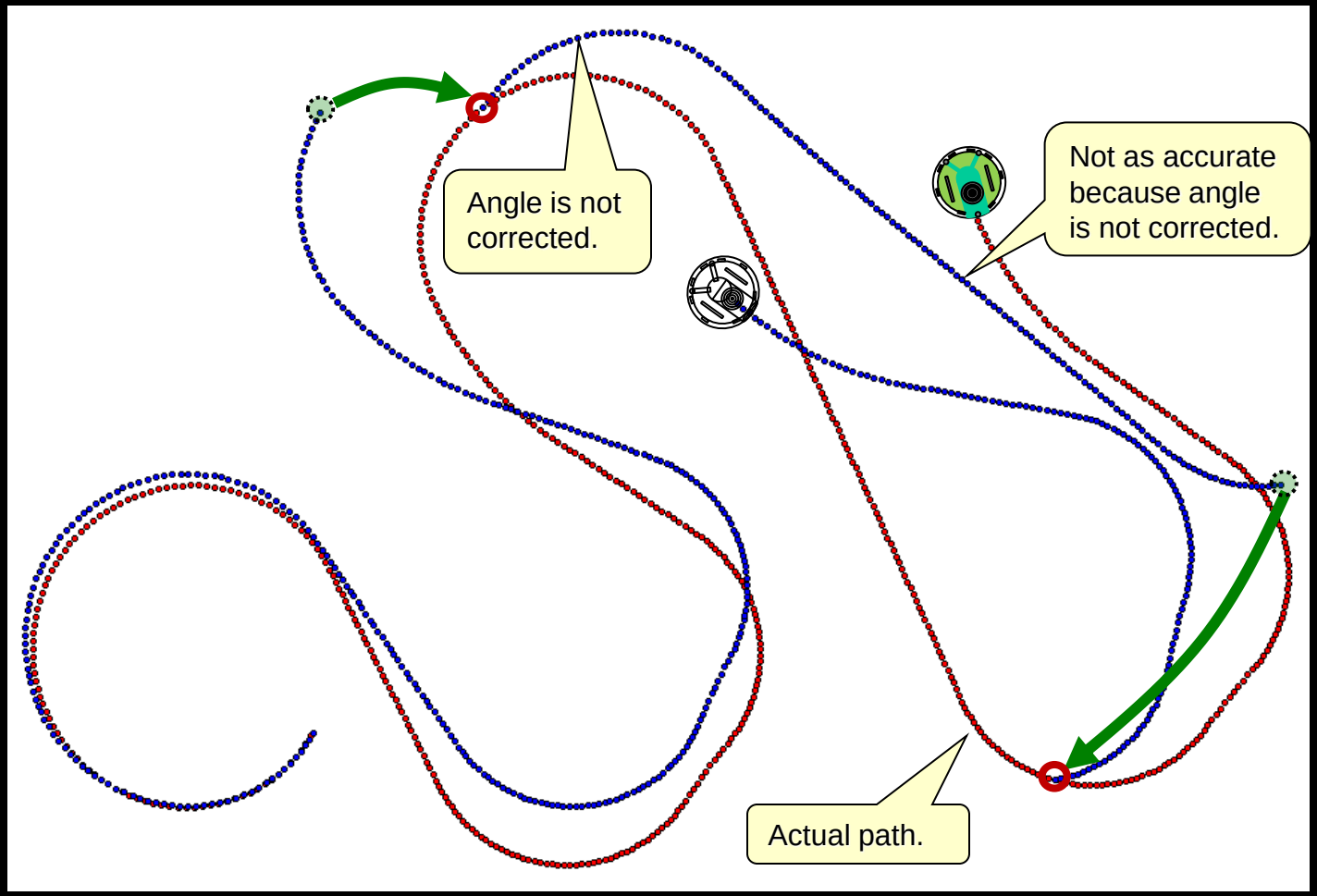
Odometry Correction – Full GPS

- If exact GPS position is given, estimate can be reset:



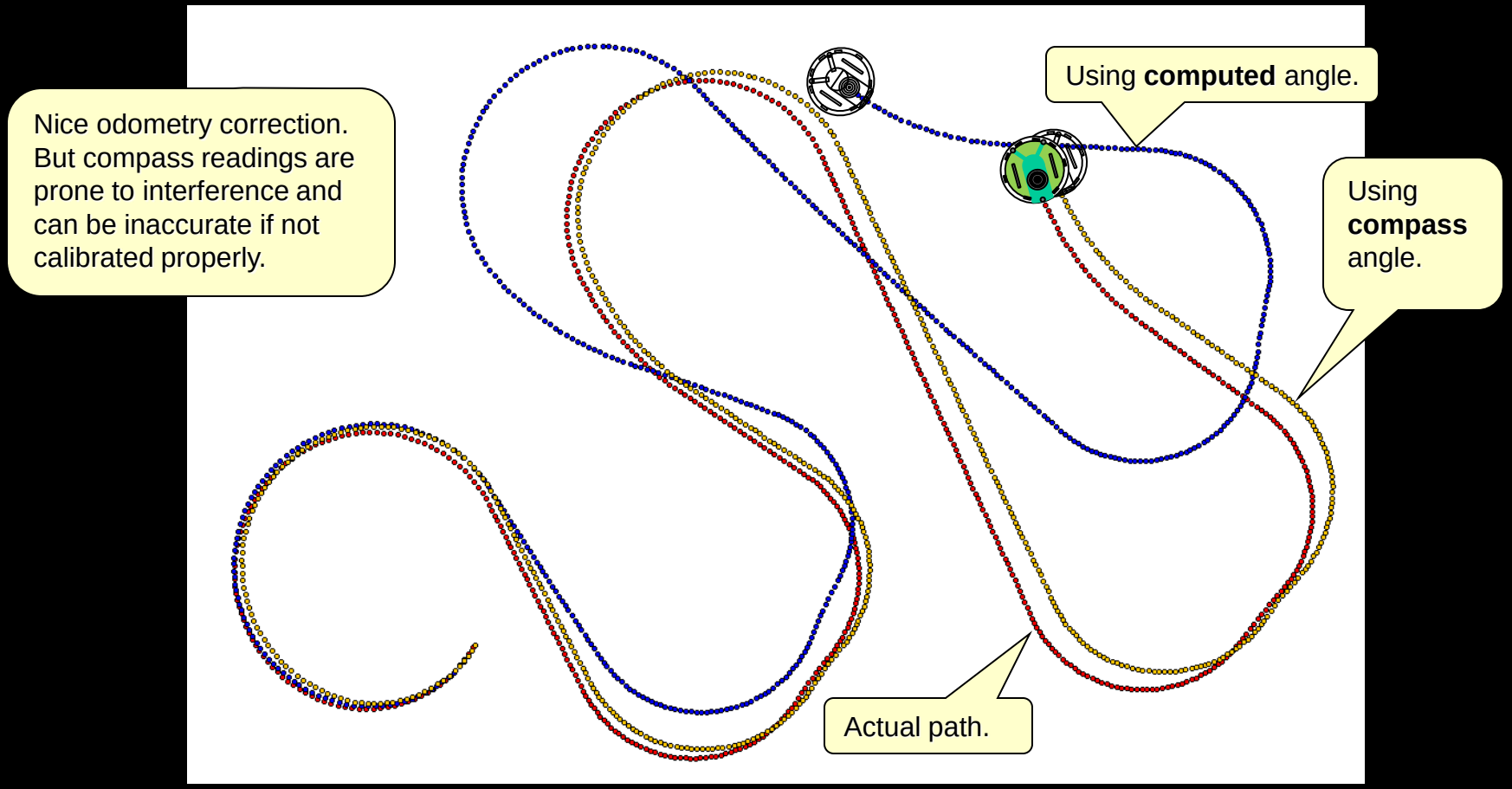
Odometry Correction – Limited GPS

- If no direction given by GPS, just update (x,y):



Odometry Correction - Compass

- If just a compass is available, errors in angles can be “fixed” if compass is accurate.



Odometry Correction - GPS

- Updating the estimate is easy.

$(x_{fk}, y_{fk}, \theta_{fk})$ = get forward kinematics estimate

First, get an estimate.

if (GPS reading is available) **then** {

$x_{fk} = x_{gps}$

$y_{fk} = y_{gps}$

Replace kinematics estimate with GPS coordinate.

$\theta_{fk} = \theta_{gps}$

If direction not available from GPS, leave this line out.

}

else if (compass is available) **then** {

$\theta_{fk} = \theta_{c}$

Replace estimate with compass reading.

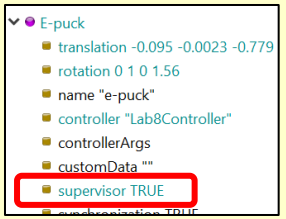
}

Getting Webot's Robot Location

- Need a way of getting actual position in order to compare.
- Webots has a Supervisor class that replaces the Robot class so that the robot can be tracked/supervised.
  - Can get the exact (x, y) location of robot in simulated world.

```
import com.cyberbotics.webots.controller.Supervisor;
import com.cyberbotics.webots.controller.Field;
import com.cyberbotics.webots.controller.Node;

Supervisor robot = new Supervisor();

// Code required for being able to get the robot's location
Node     robotNode = robot.getSelf();
Field    translationField = robotNode.getField("translation");

// while (…) {
   // Get the wheel position sensors
   double values[] = translationField.getSFVec3f();
   x =  (values[0]*100);
   y = -(values[2]*100); // Need to negate the Y value
// }
```

Using **Supervisor** instead of **Robot** now. Must set supervisor flag to TRUE in the scene tree.

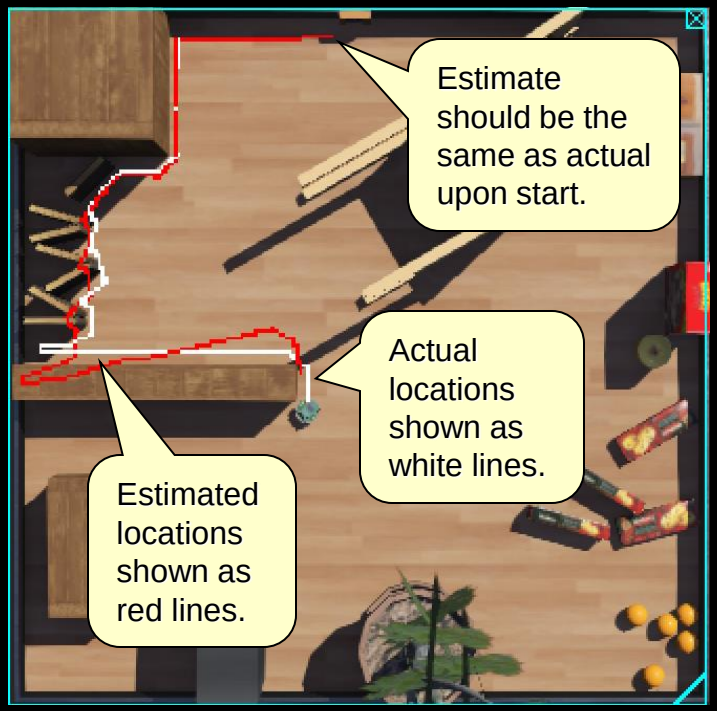


Call this each time we want the robot's location.

Convert the values into cm. Need to flip the **y** value so that origin is at bottom left.

Trace Displaying

- A simple Webots display window allows you to display the actual robot locations as well as estimated locations as the robot moves.
- Locations must come in sequence, along a boundary.



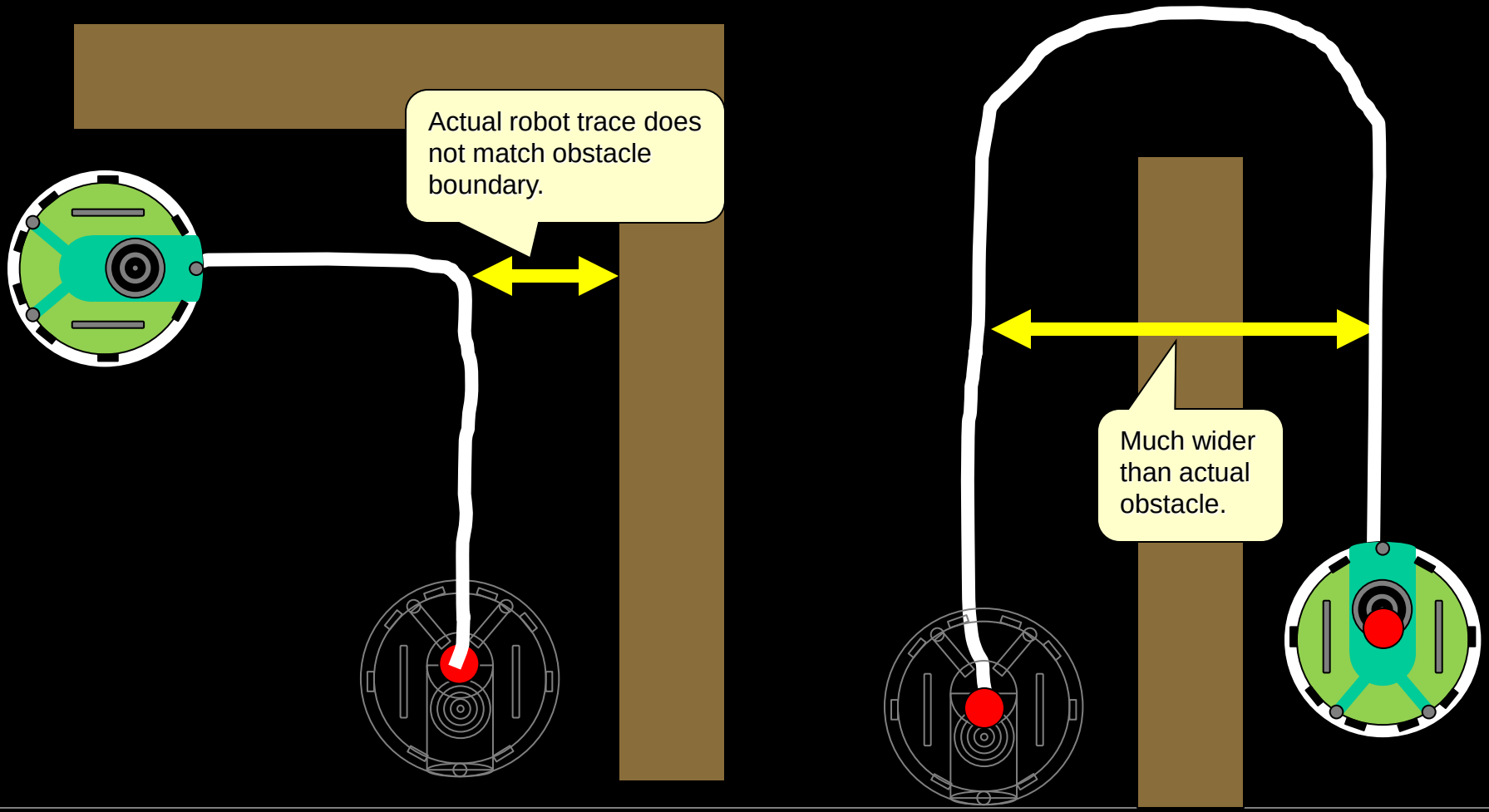
Do this once

```
TrackerApp  tracker;
tracker = new TrackerApp(robot.getDisplay("display"));


while (...) {
       ...
       tracker.addActualLocation(actualX, actualY);
       tracker.addEstimatedLocation(estimateX, estimateY);
       ...


}
```

Each time you want to display a new location, call these two functions… one to display the **actual location** of the robot, the other to display your **estimated location**. Both require integer coordinates. Don't forget to negate the **y** of the actual location (see slide 9)!

Not a Map

- This is NOT an actual MAP. It only shows locations of the center of the robot.



Compass

- A compass can be used to get an estimate of a robot's orientation with respect to a "North" heading of some sort.

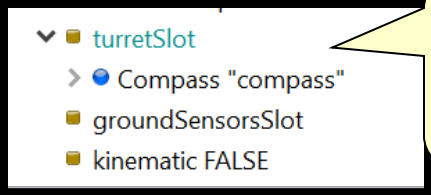
- Compasses have a lot of issues though:
  - an improperly-calibrated compass is nearly useless
    - needs to be calibrated in the environment that the robot is placed in.
  - susceptible to interference from magnetic sources, metal objects, electronics, motors, etc..

- Cannot tell when a compass is giving wrong results.

- Many compasses are inaccurate in practice and so they are not often used due to the above reasons.

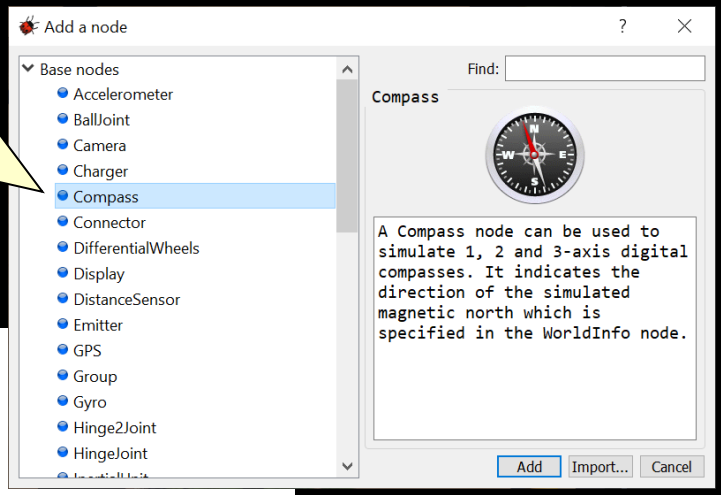
Webots Compass

- The standard e-puck does not come with a compass. We need to add one to the turretSlot.



1. Double-click **turretSlot** under the **E-puck** in the scene tree.

2. Add the **Compass** from the **Base nodes** in dialog box that appears.




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

// Get Compass sensor
Compass  compass = robot.getCompass("compass");
compass.enable(timeStep);

while (…) {
      double compassReadings[] = compass.getValues();
      double rad = Math.atan2(compassReadings[0], compassReadings[1]);
      double bearing = (rad - Math.PI/2) / Math.PI * 180.0;

      if (bearing > 180)
         bearing = 360 - bearing;
      if (bearing < -180)
         bearing = 360 + bearing;
}
```

Do this once.

It is a little bit of work to get the compass reading in degrees.

Adjust the angle so that it is always within the range of **-180°** to **180°**.

Start the Lab …