Position Estimation

Chapter 5

Objectives

- Investigate different mechanics of motion
  - We will focus on wheeled robots.
- Understand how to determine and control a robot's position and orientation within its environment by using:
  - Odometry
  - Grid Tracking
  - Active Beacons
- Learn how to program the PropBot to do position estimation in the lab.

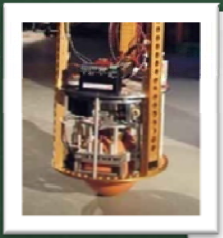
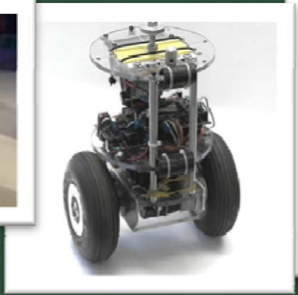
What's in Here ?

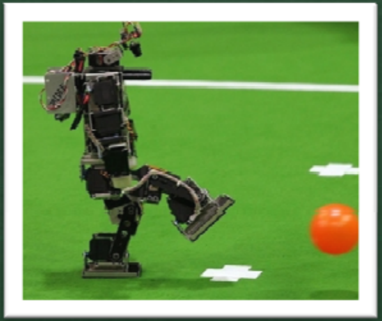
- Locomotion
  - Types and Issues
  - Wheeled Robots: Concerns and Design
  - Approaches to Position Estimation
- Odometry and Kinematics
  - Forward Kinematics
  - Inverse Kinematics
  - Estimating Speed and Orientation
- PropBot Odometry
  - PropBot Encoders
  - PropBot Forward Kinematics
  - PropBot Inverse Kinematics
  - PropBot Path Improvement Technique
- Grid Tracking
- Position Estimation Using Active Beacons
  - Beacon Triangulation (Lateration, Angulation)

Locomotion

Types of Land Locomotion

- Various ways for a robot to move:
  - Roll
    - 1, 2, 4 … wheels
    - Tank tracks
  - Walk
    - 2, 4 or 6 legs
  - Hop/Gallop/Run
    - 1, 2 or 4 legs



Locomotion – Key Issues

- **Stability**
  - # and geometry of contact points
  - Center of gravity
  - Static vs. dynamic energy
  - Terrain inclination
- **Contact Characteristics**
  - Shape / Angle of contact
  - Friction
- **Environment**
  - Structure
  - Surface (e.g., water, air, soft/hard ground)

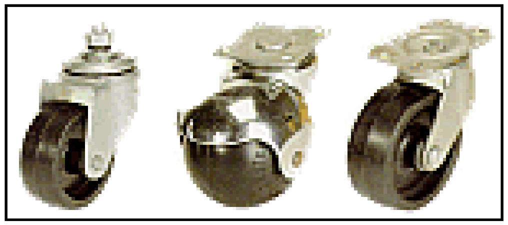
Typical Concern For Wheeled Robots In Indoor Environments

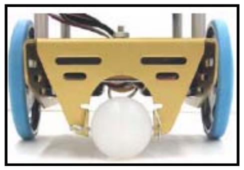
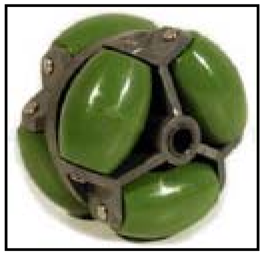
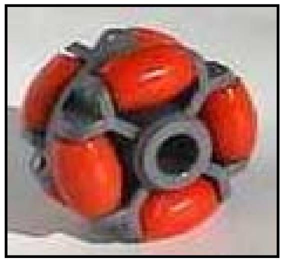
Wheeled Robot Concerns

- Stability
  - Not usually a problem, 3 wheels enough.
  - 4 wheels or more require suspension system.
- More important concerns:
  - Traction
    - Wheel slippage can lead to positioning errors.
  - Maneuverability
    - Robot may not be able to turn sharp enough.
  - Control
    - Configuration may not allow sufficient control of robot's speed, causing overshoot or collision.

Wheel Designs

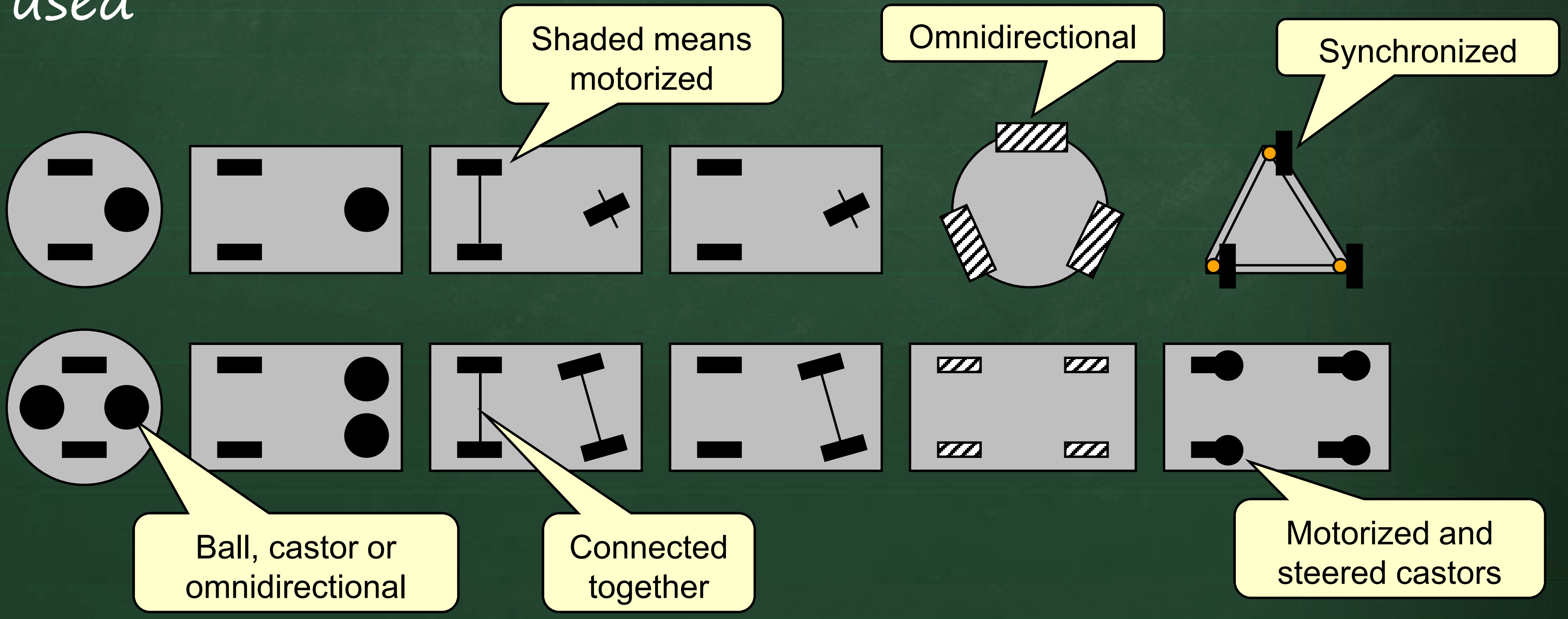
- Standard
  - often used for drive
  - sometimes used for steering
- Castor
  - used for balancing, not controlled
  - problems when changing direction
- Ball
  - no direction change problem
  - used for balancing, not controlled
- Omni-Directional
  - Less friction in "side" directions



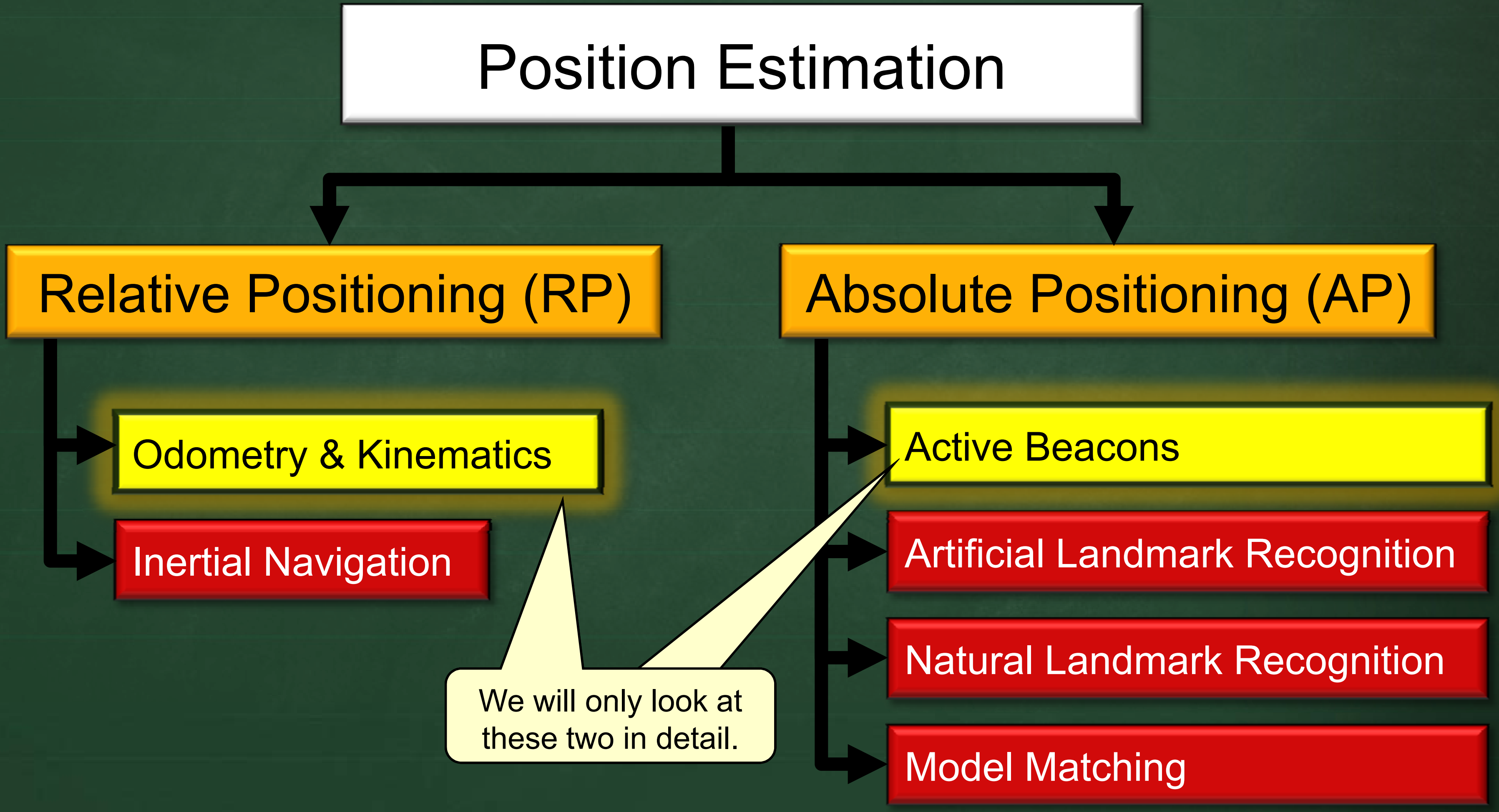




Wheel Geometry

- Choice of wheels depends on where they are placed on the robot
- Choosing geometry depends on where robot will be used



Two Main Solutions

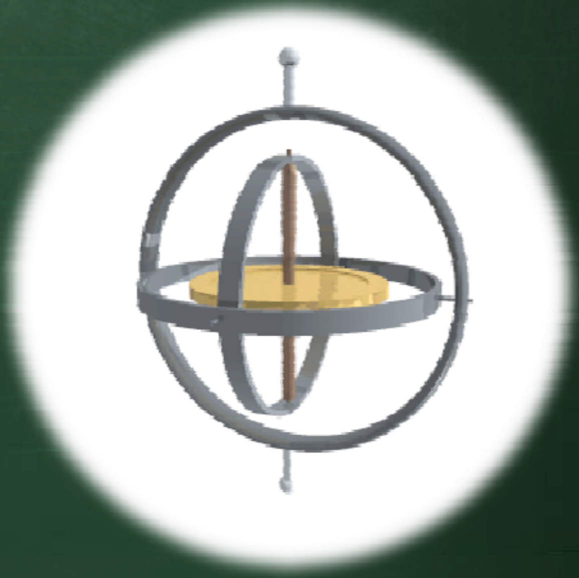

Position Estimation
Relative Positioning (RP)
Absolute Positioning (AP)
Odometry & Kinematics
Inertial Navigation
Active Beacons
Artificial Landmark Recognition
Natural Landmark Recognition
Model Matching
We will only look at these two in detail.

RP – Odometry and Kinematics

- Given wheel velocities at any given time, can compute position/orientation for any future time.
- Advantages
  - + self-contained
  - + can give positions anywhere along curved paths
  - + always gives "estimate" of position
- Disadvantages
  - – Requires accurate measurement of wheel velocities over time, including measuring acceleration and deceleration.
  - – position error grows over time

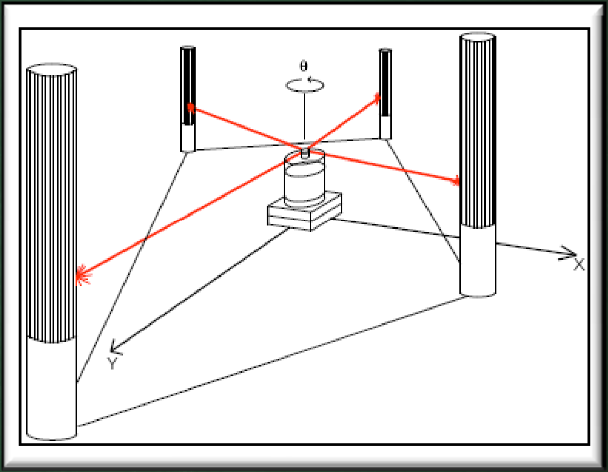
RP - Inertial Navigation

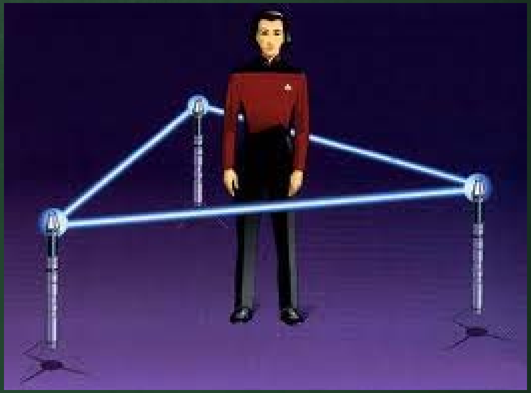
- Use gyroscopes (fiber-optic gyros or laser gyros) and accelerometers to measure rate of rotation or acceleration.
- Measurements are integrated to obtain position
- Advantages
  - + self-contained
  - + always gives "estimate" of position
- Disadvantages
  - – position error grows over time
  - – expensive sensors



AP – Active Beacons

- Uses 3 or more fixed-in-place (i.e., known sites) active (i.e., they transmit usually light or radio frequencies) beacons.
- Measures angles of incidence to each beacon.
- Advantages
  - + absolute position obtainable
  - + no accumulative errors
- Disadvantages
  - – not self-contained, fails if beacons become inactive





AP – Artificial Landmark Recognition

- Artificial landmarks placed at known locations in environment (need 3 or more visible).

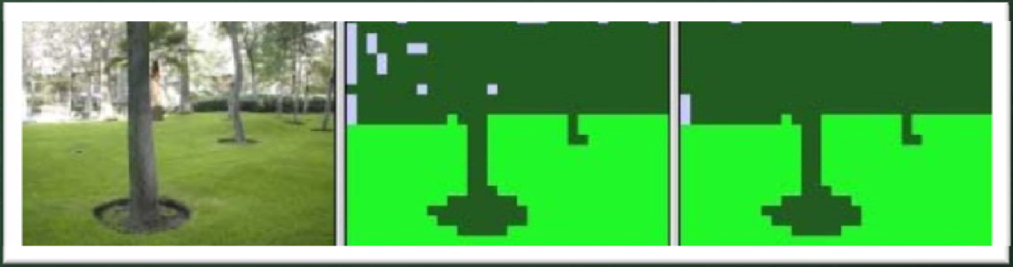
- Advantages
  - + landmarks can be designed for optimal detectability
  - + bounded position errors

Passive Landmark
IR Projector

- Disadvantages
  - – requires alteration of environment
  - – not always possible to detect landmarks
  - – landmarks may be shapes (as opposed to points), requiring measurement of geometric features

AP – Natural Landmark Recognition

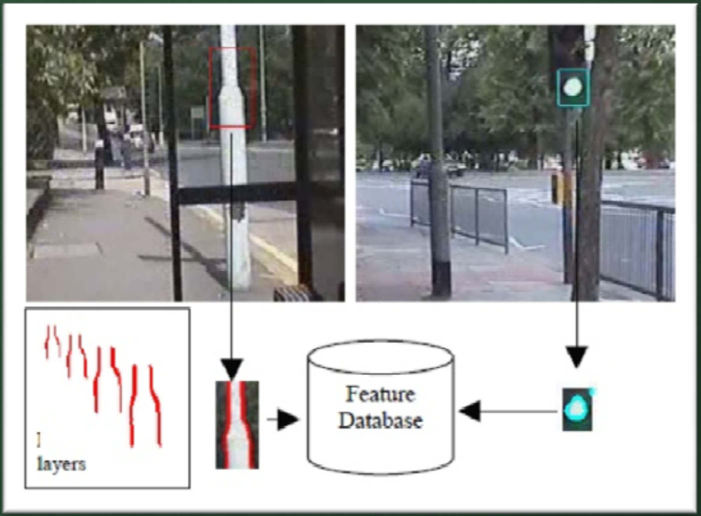
- Detect natural fixed landmarks in environment
- Determine position based on position from such landmarks



- Advantages
  - + does not require alteration of environment
- Disadvantages
  - – environment must be known in advance
  - – more difficult to detect than artificial landmarks

AP – Model Matching

- Match sensor readings with pre-known model (i.e., map) of environment

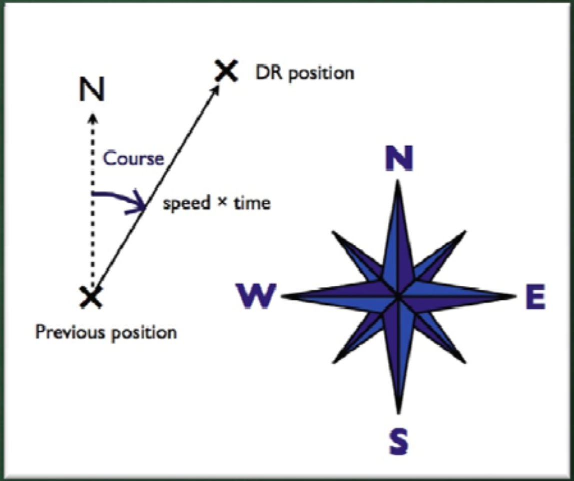


- Models can be geometric and/or topological
- Advantages
  + almost no error in position when match is found
- Disadvantages
  – sometimes impossible to determine position in models with much symmetry

Odometry and Kinematics

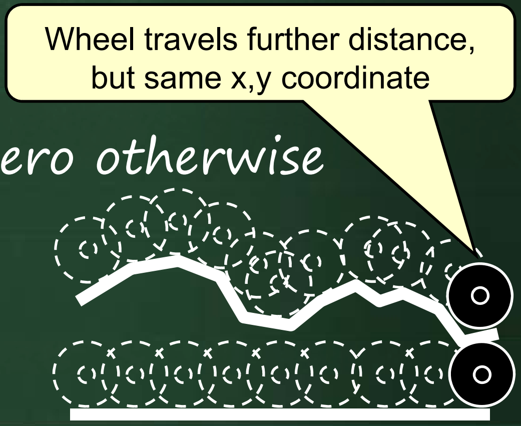
Odometry

- Odometry is a means of implementing *Dead Reckoning*:
  - A way of determining a robot's position based on previous known position information given a specific course heading and velocity.
  - Used by most mobile robot systems
  - Errors accumulate over time as robot moves due to uncertainty in measurements. Periodically requires error measurement to be "fixed" or reset (usually from external sources) in order to be useful
- Meant for short distance measurements.



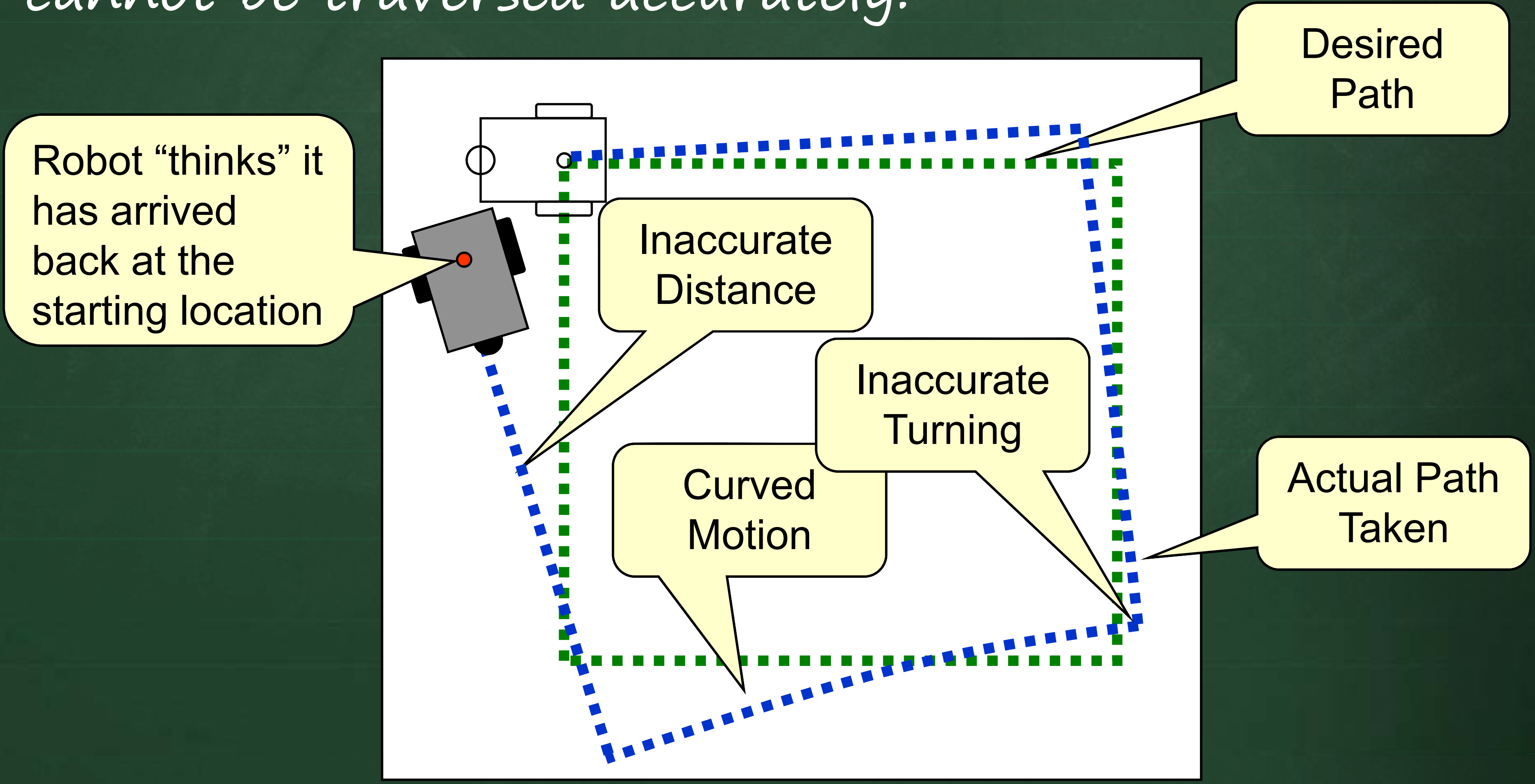
Odometry

- Errors can creep-in due to:
  - *Imprecise measurements*
    - Actual speed and turn angles not measured accurately
  - *Inaccurate control model*
    - Wheels are not infinitely thin and do not make contact with the ground surface at a single point
    - Wheels are not exactly the same size with axles aligned perfectly
  - *Immeasurable physical characteristics*
    - Friction is not infinite in rolling direction and zero otherwise
    - Wheels wobble slightly and skid during turns
    - Surface is not perfectly smooth and hard



Odometry

- As a result of these error factors, a simple path cannot be traversed accurately.



Odometry

- The robot may become "lost" quite quickly.
- Theoretically, we can calculate the actual robot's position as long as:
  - the robot's structure is well known
  - the robot's wheel acceleration, deceleration and velocities can be accurately measured

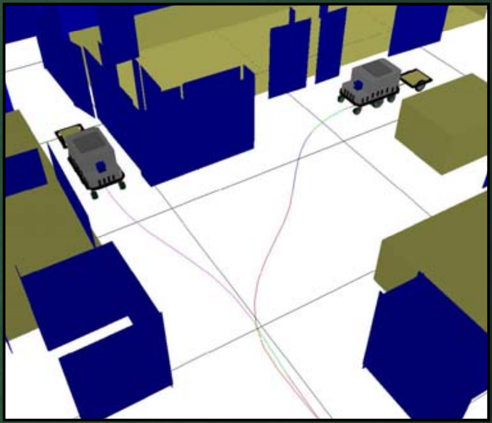
- Hence, we can detect our errors during travel and adjust the path accordingly
- This is done through ***kinematics***.

Kinematics

- **Kinematics** is

  a branch of physics that deals with aspects of motion apart from considerations of mass and force

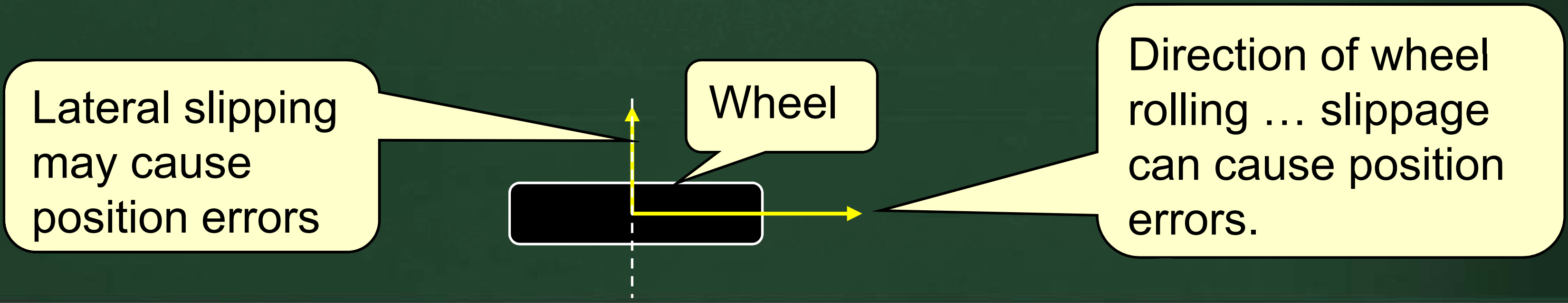
- Mobile robot's ***workspace*** defines the range of possible ***poses*** (i.e., positions) that the robot can achieve



- Mobile robot's ***controllability*** defines the possible paths and trajectories in its workspace.
- Must understand the contribution that each wheel provides for motion

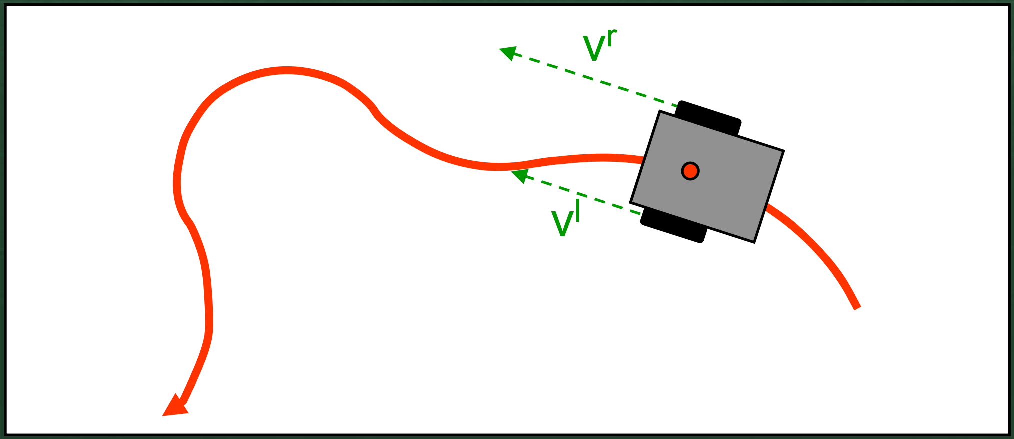
Kinematics

- Estimating a self-contained autonomous robot's position is difficult:
  - No direct way to measure from sensors
  - Must integrate the motion of the robot over time
  - Inaccuracies of motion estimation (e.g., wheel slippage)
- Must define a robot ***model*** based on its geometry
  - Each wheel contributes to motion and also imposes constraints (e.g., inability to move/skid laterally)


Lateral slipping may cause position errors
Wheel
Direction of wheel rolling … slippage can cause position errors.

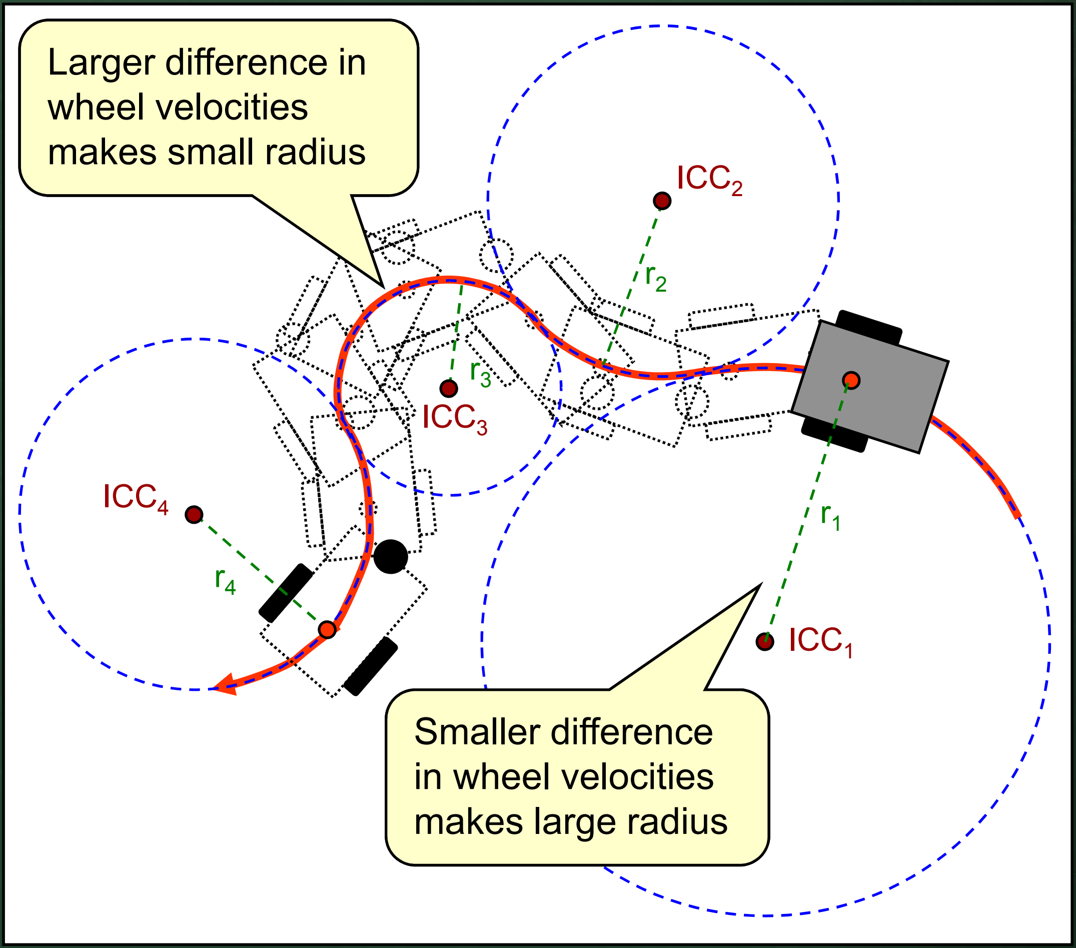
Kinematics

- Consider a differential drive robot like the PropBot
- At any instance in time both left and right wheels have their own velocities v^l and v^r
- Robot forms curves in its workspace depending on these velocities



Kinematics

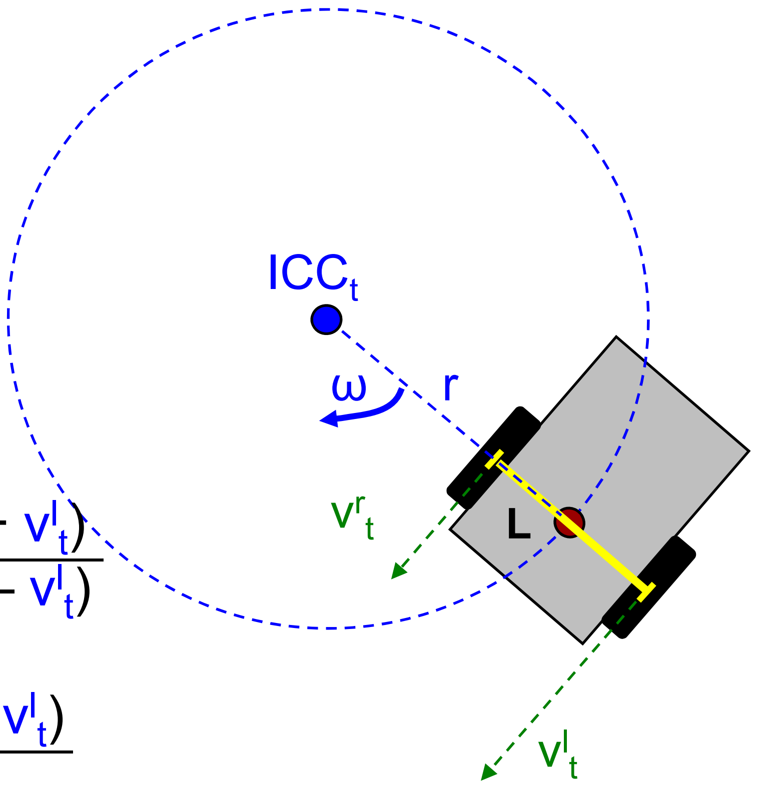
- The *instantaneous center of curvature* (ICC) is the point around which each wheel of the robot makes a circular course.
- ICC changes over time as a function of the individual wheel velocities



Larger difference in wheel velocities makes small radius
ICC$_2$
r_2
r_3
ICC$_3$
ICC$_4$
r_4
r_1
ICC$_1$
Smaller difference in wheel velocities makes large radius

Kinematics

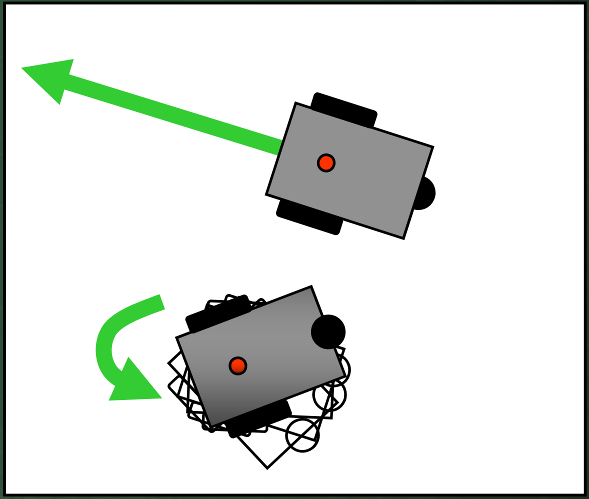
- Assume that at each instance of time, the robot is following a curve around some ICC_t with radius r at angular rate ω with left and right wheel velocities v^l_t and v^r_t, respectively.



$$r = \frac{L(v^r_t + v^l_t)}{2(v^r_t - v^l_t)}$$

$$\omega = \frac{(v^r_t - v^l_t)}{L}$$

Kinematics

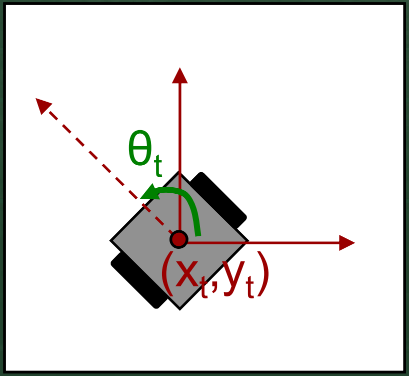


- There are three cases:
  - When $v^l_t = v^r_t$, then r is infinite and the robot moves in a straight line.
  - When $v^l_t = -v^r_t$, then r is zero and the robot spins (i.e., rotates in place).
  - In all other cases then r is finite and non-zero and the robot follows a curved trajectory about a point which is distance r away from the robot's center
- Differential drive robots are very sensitive to the velocity differences between the two wheels … making it hard to move in a perfectly straight line.

Kinematics

- A robot's *pose* is its position (in terms of location and orientation) with respect to some global coordinate system
  - pose p_t at time t is represented by vector

$$p_t = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix}$$

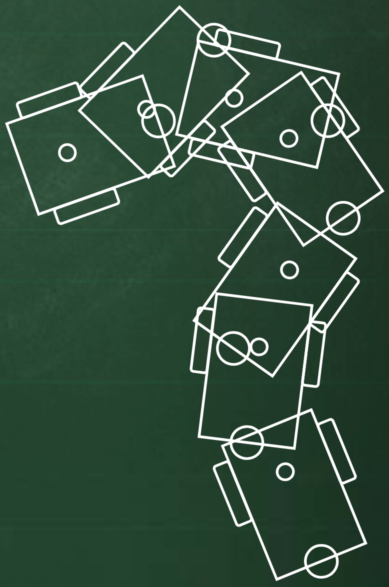


Forward Kinematics

- Consider the *forward kinematics* problem:
  - Given some control parameters (e.g., wheel velocities), determine the poses which are possible for the robot.

Given: v^l_t, v^r_t and p_t

Find: $p_{t+\delta}$



- Therefore, $p_{t+\delta}$ is defined recursively as a function of the wheel velocities:

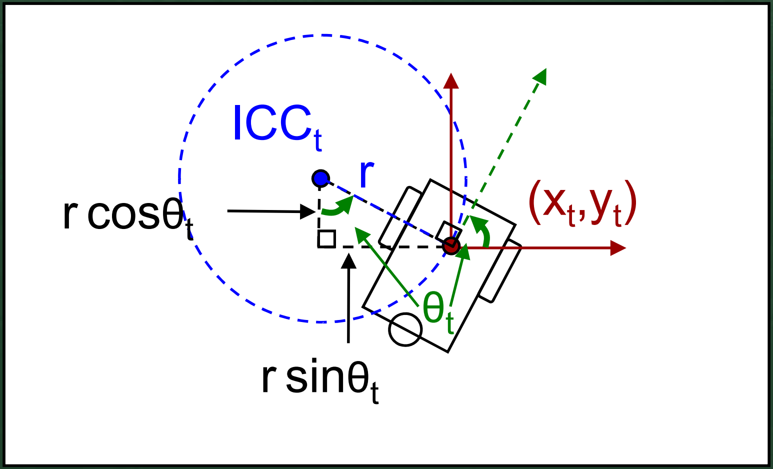
$$p_{t+\delta} = F(v^l_t, v^r_t)\ p_t$$

Forward Kinematics

- To solve the forward kinematics problem, we must determine $ICC_t = (ICC^x_t, ICC^y_t)$
  - (i.e., the curve that the robot is traveling on at time t)

$$ICC_t = (x_t - r \cdot \sin\theta_t \; , \; y_t + r \cdot \cos\theta_t)$$

$$\text{where } r = L(V^r_t + V^l_t) \,/\, (2(V^r_t - V^l_t))$$



Forward Kinematics

- Hence at time $t + \delta$, the robot's pose is:

Standard matrix for rotation about the Z-axis.

$$p_{t+\delta} = \begin{bmatrix} x_{t+\delta} \\ y_{t+\delta} \\ \theta_{t+\delta} \end{bmatrix} = \begin{bmatrix} \cos(\omega\delta) & -\sin(\omega\delta) & 0 \\ \sin(\omega\delta) & \cos(\omega\delta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t - ICC^x_t \\ y_t - ICC^y_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} ICC^x_t \\ ICC^y_t \\ \omega\delta \end{bmatrix}$$

- Since $ICC^x_t = x_t - r\cdot\sin\theta_t$ and $ICC^y_t = y_t + r\cdot\cos\theta_t$ then:

$$p_{t+\delta} = \begin{bmatrix} r\cdot\cos\omega\delta\cdot\sin\theta_t + r\cdot\cos\theta_t\cdot\sin\omega\delta + x_t - r\cdot\sin\theta_t \\ r\cdot\sin\omega\delta\cdot\sin\theta_t - r\cdot\cos\theta_t\cdot\cos\omega\delta + y_t + r\cdot\cos\theta_t \\ \theta_t + \omega\delta \end{bmatrix}$$

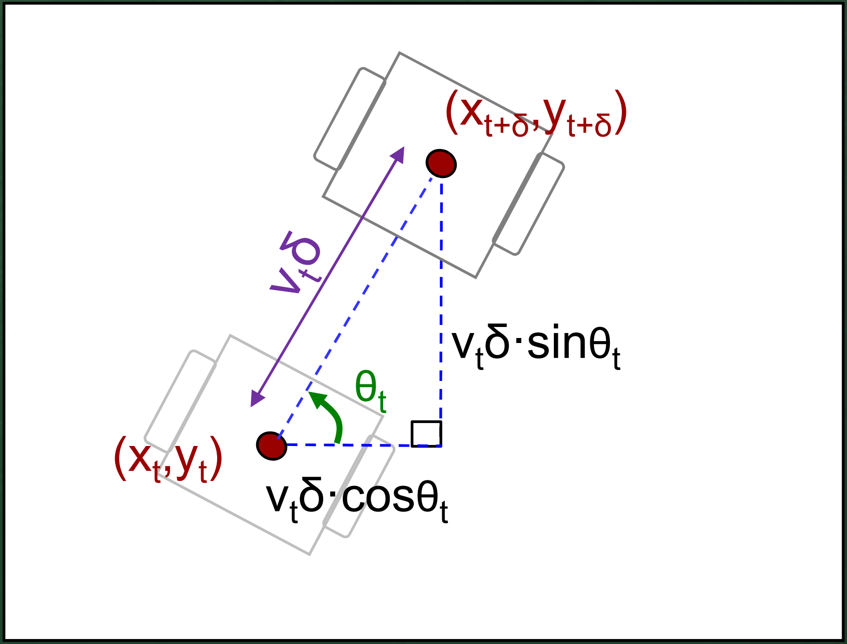
Forward Kinematics

- In the special case where $v^l_t = v^r_t$, we cannot use this equation since $r = \infty$.
  - need to ignore ICC
- Easy to show that when $v_t = v^r_t = v^l_t$ then:

$$x_{t+\delta} = x_t + v_t\delta \cos\theta_t$$

$$y_{t+\delta} = y_t + v_t\delta \sin\theta_t$$

$$\theta_{t+\delta} = \theta_t$$



$(x_{t+\delta},y_{t+\delta})$
$v_t\delta$
$v_t\delta\cdot\sin\theta_t$
θ_t
(x_t,y_t)
$v_t\delta\cdot\cos\theta_t$

Forward Kinematics - Example

Example: Differential steering robot with L = 10.6cm

This value is different for our lab robots which have a value of **L = 8.9cm**

- Robot starts at position $(x_o, y_o) = (20_{cm}, 20_{cm})$ and orientation $\theta = 0°$ at time $t = 0_{sec}$.
- Robot sets both wheels to speed of $2_{cm/sec}$ and moves for 10_{sec}. Where is the robot at $t = 10_{sec}$?
  - Assume negligible acceleration (reasonable for slow speeds)

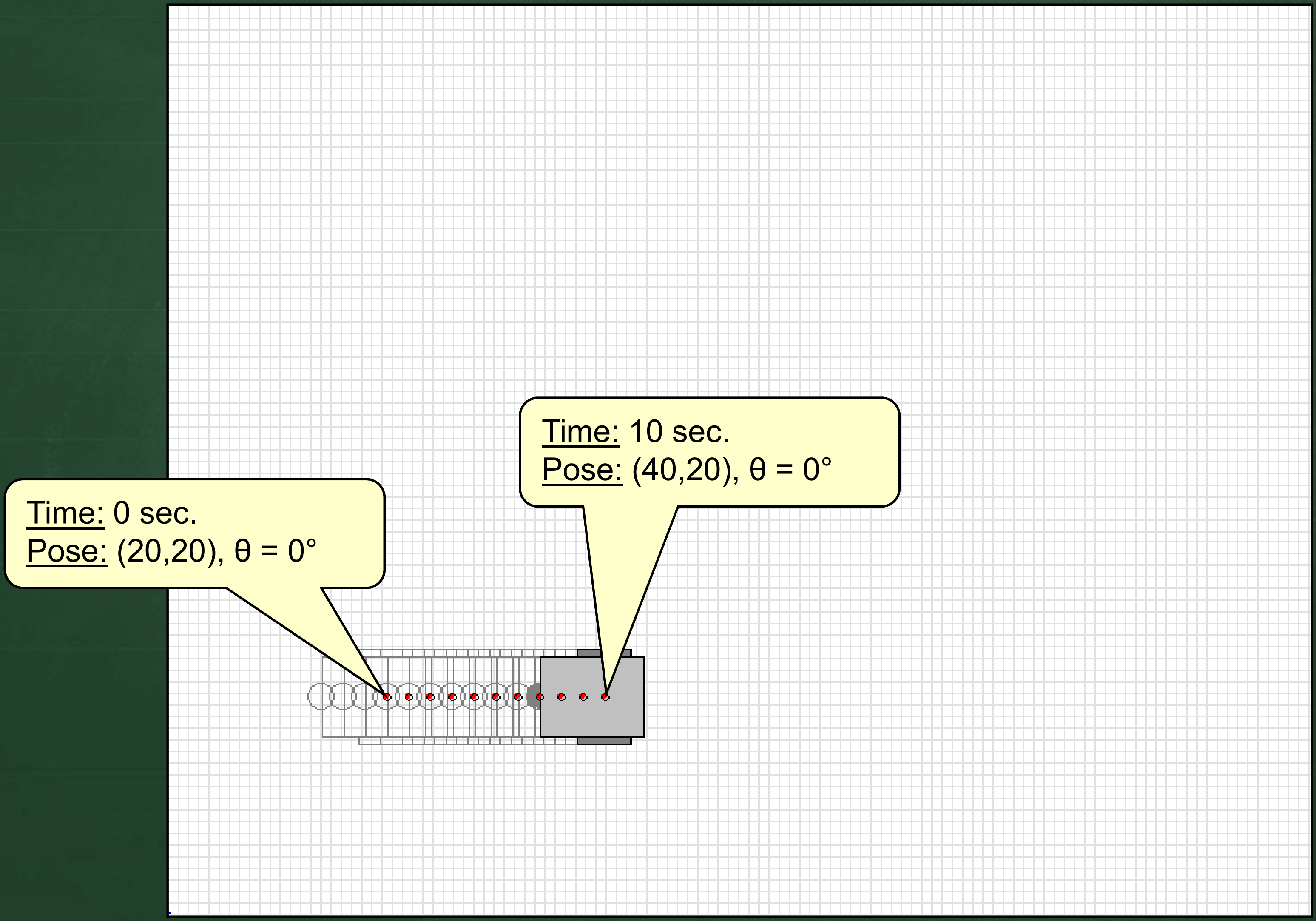
    $r = \infty,\ \omega = 0,\ \delta = 10_{sec}$

    $x_{t+\delta} = x_t + v_t\delta \cos\theta_t = 20_{cm} + 2_{cm/sec} * 10_{sec} * 1 = \underline{40}_{cm}$

    $y_{t+\delta} = y_t + v_t\delta \sin\theta_t = 20_{cm} + 2_{cm/sec} * 10_{sec} * 0 = \underline{20}_{cm}$

    $\theta_{t+\delta} = \theta_t = \underline{0}°$

Forward Kinematics – Example



Forward Kinematics – Example

- Now robot sets the right wheel to $3_{cm/sec}$ and the left wheel to $2_{cm/sec}$ and does this for 10_{sec} more.
  - Where is the robot at $t = 20_{sec}$? Here $\delta = 10_{sec}$

    $r = L(v^r_t + v^l_t) / (2(v^r_t - v^l_t)) = 10.6_{cm} * 5_{cm/sec} / (2 * 1_{cm/sec}) = \underline{26.5}_{cm}$

    $\omega = (v^r_t - v^l_t) / L = (1_{cm/sec}) / 10.6_{cm} = \underline{0.0943396}_{rad/sec}$

    $x_{t+\delta} = r{\cdot}\cos(\omega\delta)\sin\theta_t + r{\cdot}\cos\theta_t \sin(\omega\delta) + x_t - r{\cdot}\sin\theta_t$

    $= 26.5 * 0.587 * 0 + 26.5 * 1 * 0.810 + 40 - 26.5 * 0$

    $= 0 + 21.465 + 40 - 0 = \underline{61.465}_{cm}$

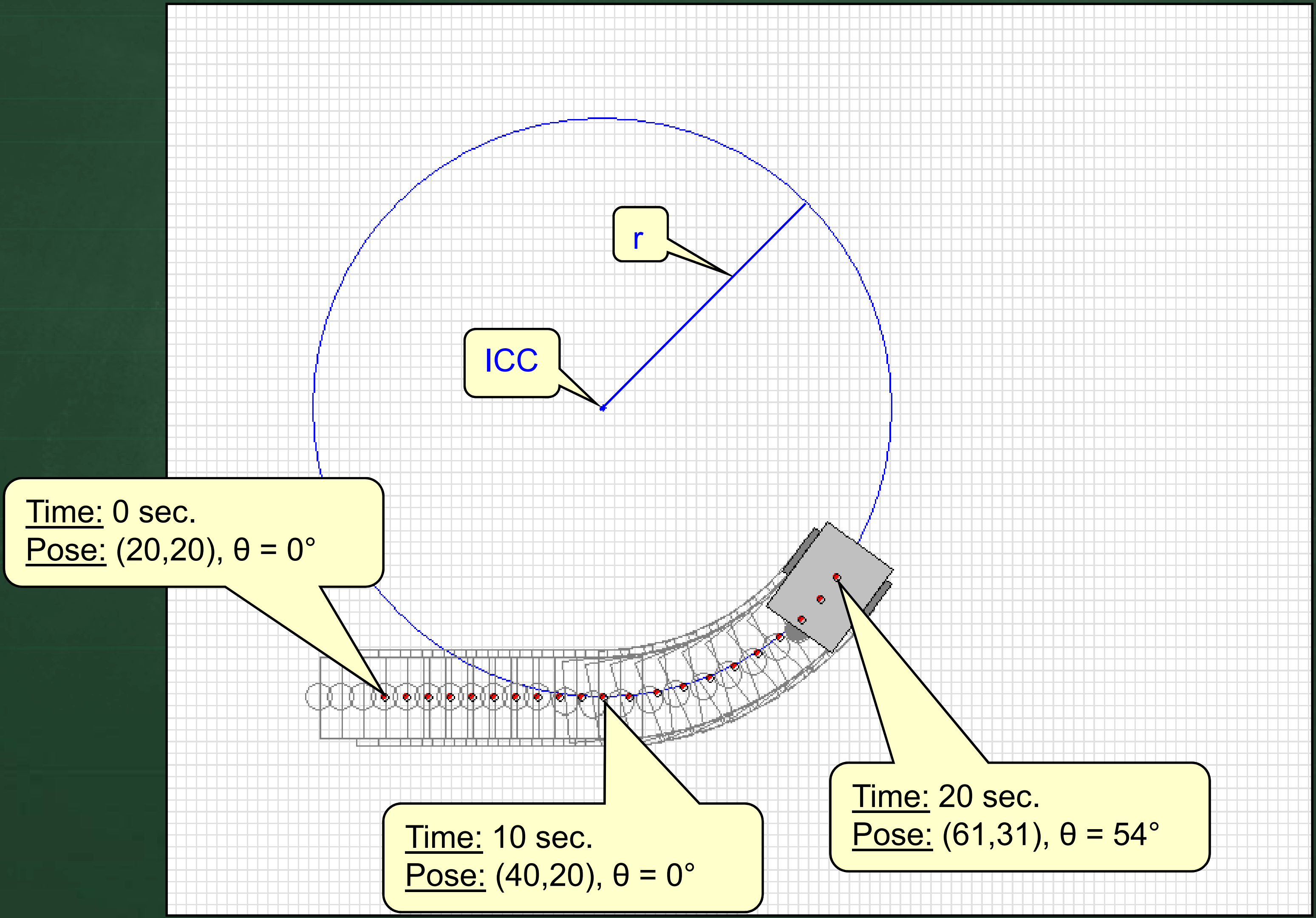
    $y_{t+\delta} = r{\cdot}\sin(\omega\delta)\sin\theta_t - r{\cdot}\cos\theta_t \cos(\omega\delta) + y_t + r{\cdot}\cos\theta_t$

    $= 26.5 * 0.810 * 0 - 26.5 * 1 * 0.58704 + 20 + 26.5 * 1$

    $= 0 \; - 15.55 + 20 + 26.5 = \underline{30.95}_{cm}$

    $\theta_{t+\delta} = \theta_t + \omega\delta = 0° + 0.0943396_{rad/sec} * 10_{sec} * 180°/\pi = \underline{54}°$

Forward Kinematics – Example



Forward Kinematics - Example

- Now robot sets the right wheel to $2_{cm/sec}$ and the left wheel to $-2_{cm/sec}$ and does this for 5_{sec} more.
  - Where is the robot at $t = 25_{sec}$? Here $\delta = 5_{sec}$

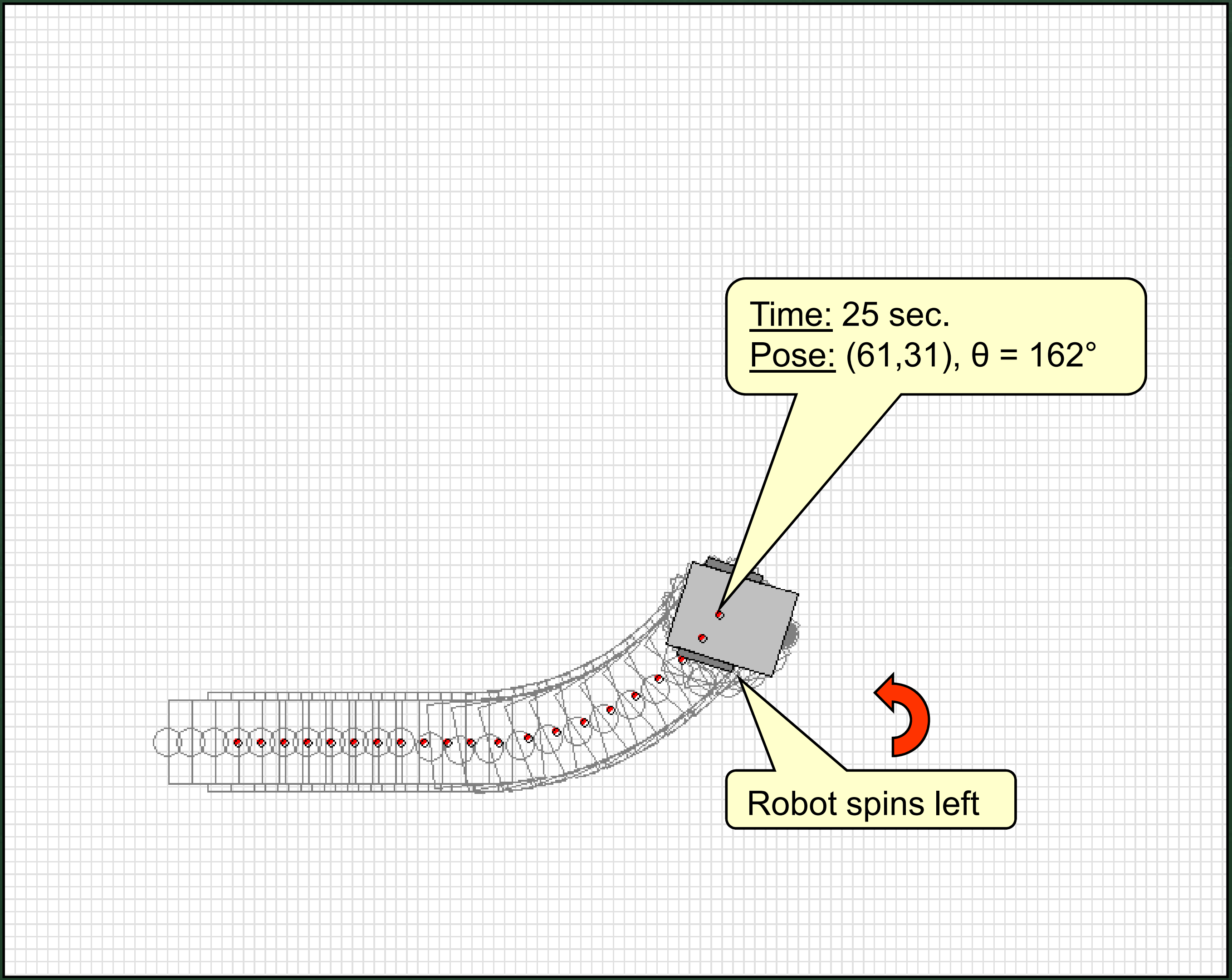
$\omega = (v^r_t - v^l_t) / L = (2_{cm/sec} - -2_{cm/sec}) / 10.6_{cm} = \underline{0.377358}_{rad/sec}$

$x_{t+\delta} = \underline{61.45}_{cm}$

$y_{t+\delta} = \underline{30.94}_{cm}$

$\theta_{t+\delta} = \theta_t + \omega\delta = 54° + 0.377358_{rad/sec} * 5_{sec} * 180°/\pi = \underline{162}°$

Forward Kinematics – Example



Forward Kinematics - Example

- Now robot sets the right wheel to $3_{cm/sec}$ and the left wheel to $3.5_{cm/sec}$ and does this for 15_{sec}.
  - Where is the robot at $t = 40_{sec}$? Here $\delta = 15_{sec}$, $\theta_t = 162°$

$r = L(v^r_t + v^l_t) / (2(v^r_t - v^l_t)) = 10.6_{cm} * 6.5_{cm/sec} / (2 * -0.5_{cm/sec}) = \underline{-68.9}_{cm}$

$\omega = (v^r_t - v^l_t) / L = -0.5_{cm/sec} / 10.6_{cm} = \underline{-0.04717}_{rad/sec}$

$x_{t+\delta} = r\cdot\cos(\omega\delta)\sin\theta_t + r\cdot\cos\theta_t\sin(\omega\delta) + x_t - r\cdot\sin\theta_t$

$= -68.9*0.76*0.309 + -68.9*-0.951*-0.65 + 61 - -68.9*0.309$

$= -16.18 - 42.59 + 61 + 21.29 = \underline{23.52}_{cm}$

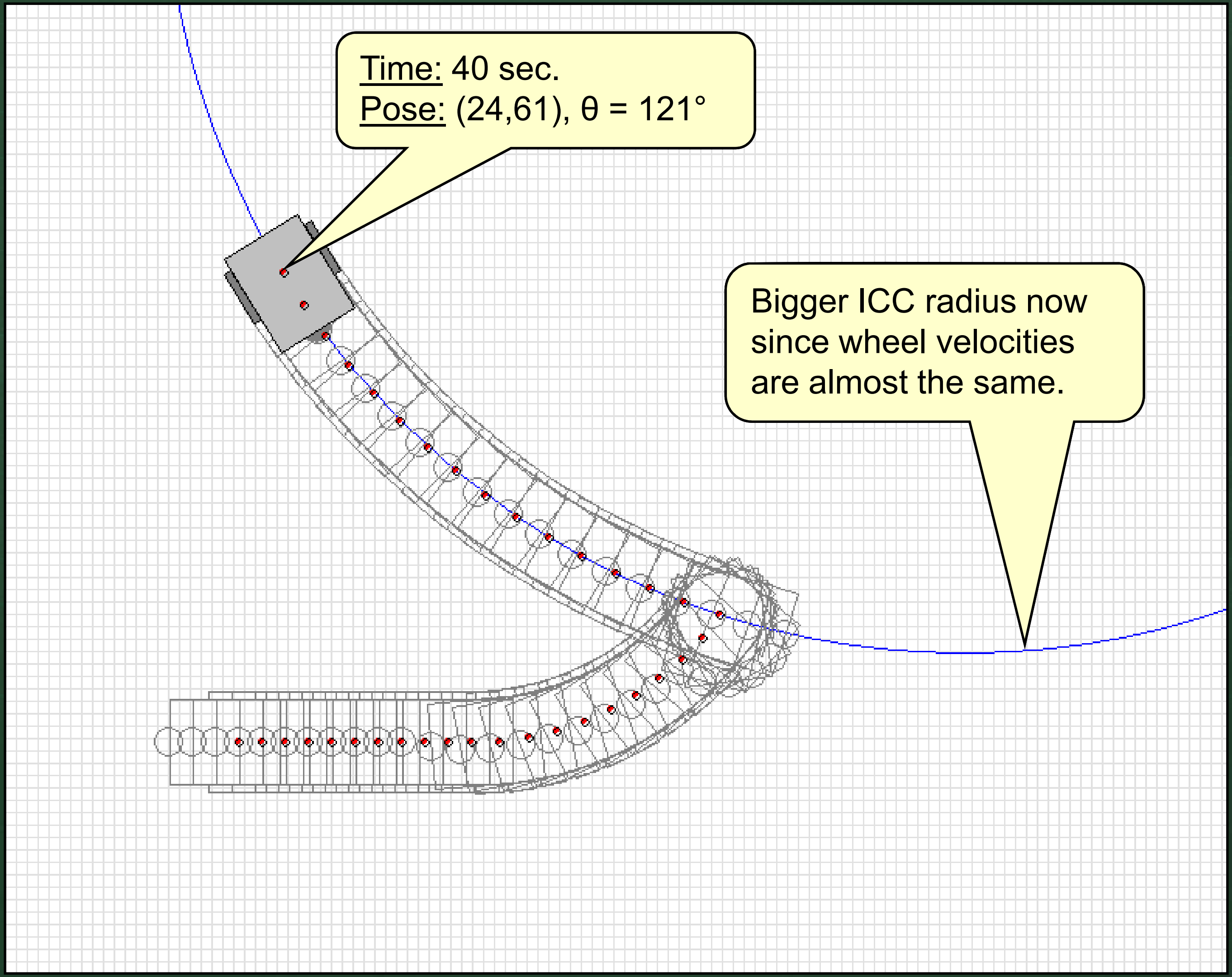
$y_{t+\delta} = r\cdot\sin(\omega\delta)\sin\theta_t - r\cdot\cos\theta_t\cos(\omega\delta) + y_t + r\cdot\cos\theta_t$

$= -68.9*-0.65*0.309 - -68.9*-0.951*0.76 + 31 + -68.9*-0.951$

$= 13.83 - 49.8 + 31 + 65.52 = \underline{60.55}_{cm}$

$\theta_{t+\delta} = \theta_t + \omega\delta = 162° - 0.04717_{rad/sec} * 15_{sec} * 180°/\pi = \underline{121}°$

Forward Kinematics – Example


Time: 40 sec.
Pose: (24,61), θ = 121°
Bigger ICC radius now since wheel velocities are almost the same.

Forward Kinematics – Example

- Finally, robot sets the right wheel to $0_{cm/sec}$ and the left wheel to $3_{cm/sec}$ and does this for 10_{sec}.
  - Where is the robot at $t = 50_{sec}$? Here $\delta = 10_{sec}$, $\theta_t = 121°$

$r = L(v^r_t + v^l_t) / (2(v^r_t - v^l_t)) = 10.6_{cm} * 3_{cm/sec} /(2*-3_{cm/sec}) = \underline{-5.3}_{cm}$

$\omega = (v^r_t - v^l_t) / L = -3_{cm/sec} / 10.6_{cm} = \underline{-0.28302}_{rad/sec}$

$x_{t+\delta} = r \cdot \cos(\omega\delta)\sin\theta_t + r \cdot \cos\theta_t \sin(\omega\delta) + x_t - r \cdot \sin\theta_t$

$= -5.3*-0.952*0.857 + -5.3*-0.515*-0.306 + 23.52 - -5.3*0.857$

$= 4.324 - 0.835 + 23.52 + 4.54 = \underline{31.5}_{cm}$

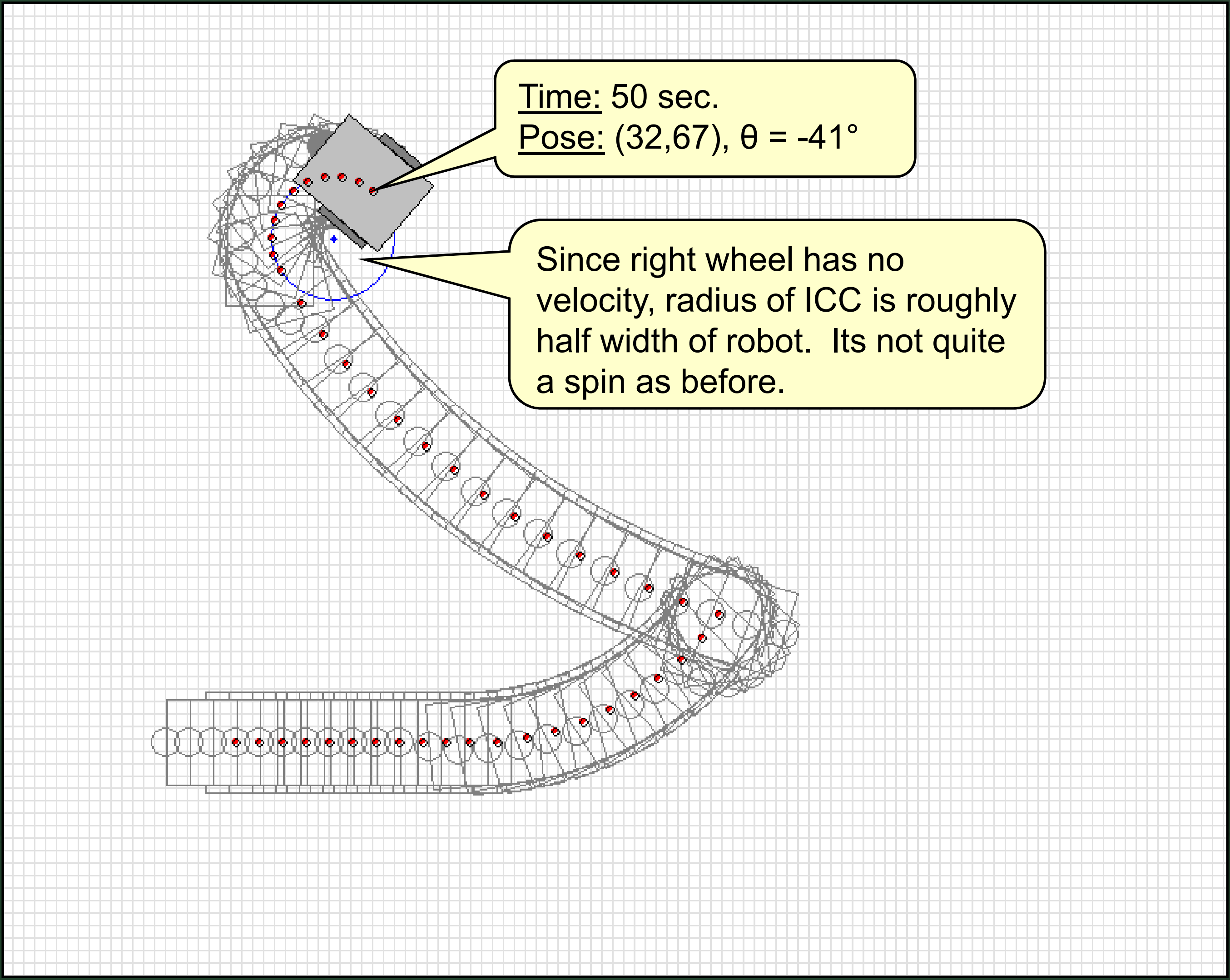
$y_{t+\delta} = r \cdot \sin(\omega\delta)\sin\theta_t - r \cdot \cos\theta_t \cos(\omega\delta) + y_t + r \cdot \cos\theta_t$

$= -5.3*-0.306*0.952 - -5.3*-0.515*-0.952 + 60.6 + -5.3*-0.515$

$= 1.54 + 2.60 + 60.6 + 2.73 = \underline{67.47}_{cm}$

$\theta_{t+\delta} = \theta_{(t)} + \omega\delta = 121° - 0.28302_{rad/sec} * 10_{sec} * 180°/\pi = \underline{-41}°$

Forward Kinematics – Example



Time: 50 sec.
Pose: (32,67), θ = -41°

Since right wheel has no velocity, radius of ICC is roughly half width of robot. Its not quite a spin as before.

Forward Kinematics

- So what about acceleration ? Approximate it …
  - Measure velocity of each motor at small intervals as robot is accelerating or decelerating
  - Apply forward kinematics to each interval (which will differ from one to the next as robot ac(de)celerates).
  - Sampling speeds at faster rates results in higher approximation accuracy.

- Will this be accurate ?
  - Nothing is accurate in robotics … always a measure of error.
  - Wheels will slip, slide sideways, and there is no way to get a precise measurement of wheel velocities.

Inverse Kinematics

- Consider the *inverse kinematics* problem:
  - Determine the control parameters (e.g., wheel velocities), that will make the robot move to a new pose from its current pose.

<u>Given:</u> p_t and $p_{t+\delta}$ <u>Find:</u> v^l_t and v^r_t

- All you have to do is solve for v^l_t and v^r_t in:

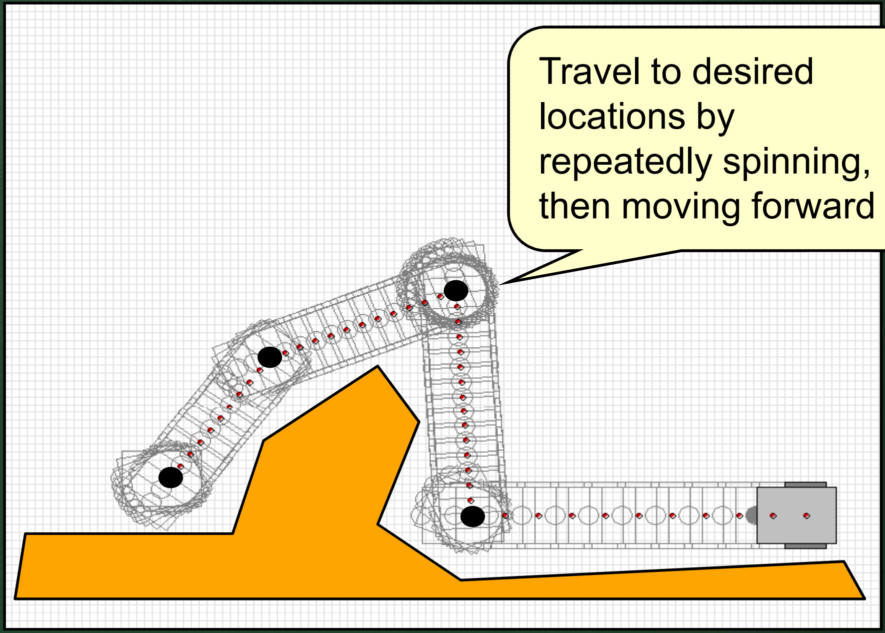
$$x_{t+\delta} = r \cdot \cos(\omega\delta)\sin\theta_t + r \cdot \cos\theta_t \sin(\omega\delta) + x_t - r \cdot \sin\theta_t$$

$$y_{t+\delta} = r \cdot \sin(\omega\delta)\sin\theta_t - r \cdot \cos\theta_t \cos(\omega\delta) + y_t + r \cdot \cos\theta_t$$

$$\theta_{t+\delta} = \theta_t + \omega\delta$$

Inverse Kinematics

- It is a difficult problem !!
  - There are too many unknowns and multiple solutions.
  - We will not solve it in this course.
- Easy way to get robot from point to point:
  - Spin in location until desired angle and then move forward


Travel to desired locations by repeatedly spinning, then moving forward

Inverse Kinematics

- Questions remain:
  - What velocities do we use for the wheels ?
  - How long do we spin or move ahead ?
- Length of time to spin depends on velocity:
  - From $\theta_{t+\delta} = \theta_t + \omega\delta$ we obtain:

    $\delta = (\theta_{t+\delta} - \theta_t) / \omega$
  - But we already know that $\omega = (v^r_t - v^l_t) / L$ and $v^l_t = -v^r_t$ so:

    $\delta = (L/2)(\theta_{t+\delta} - \theta_t) / v^r_t$ … we could have similarly fixed δ and solved for v^r_t, v^l_t
- Hence set wheels at some fixed velocity v^l_t & $-v^r_t$ for amount of time δ to achieve desired orientation.

Inverse Kinematics

- So what about moving forward ? … We can use:

$$x_{t+\delta} = x_t + v_t\delta \cos\theta_t$$

$$y_{t+\delta} = y_t + v_t\delta \sin\theta_t$$

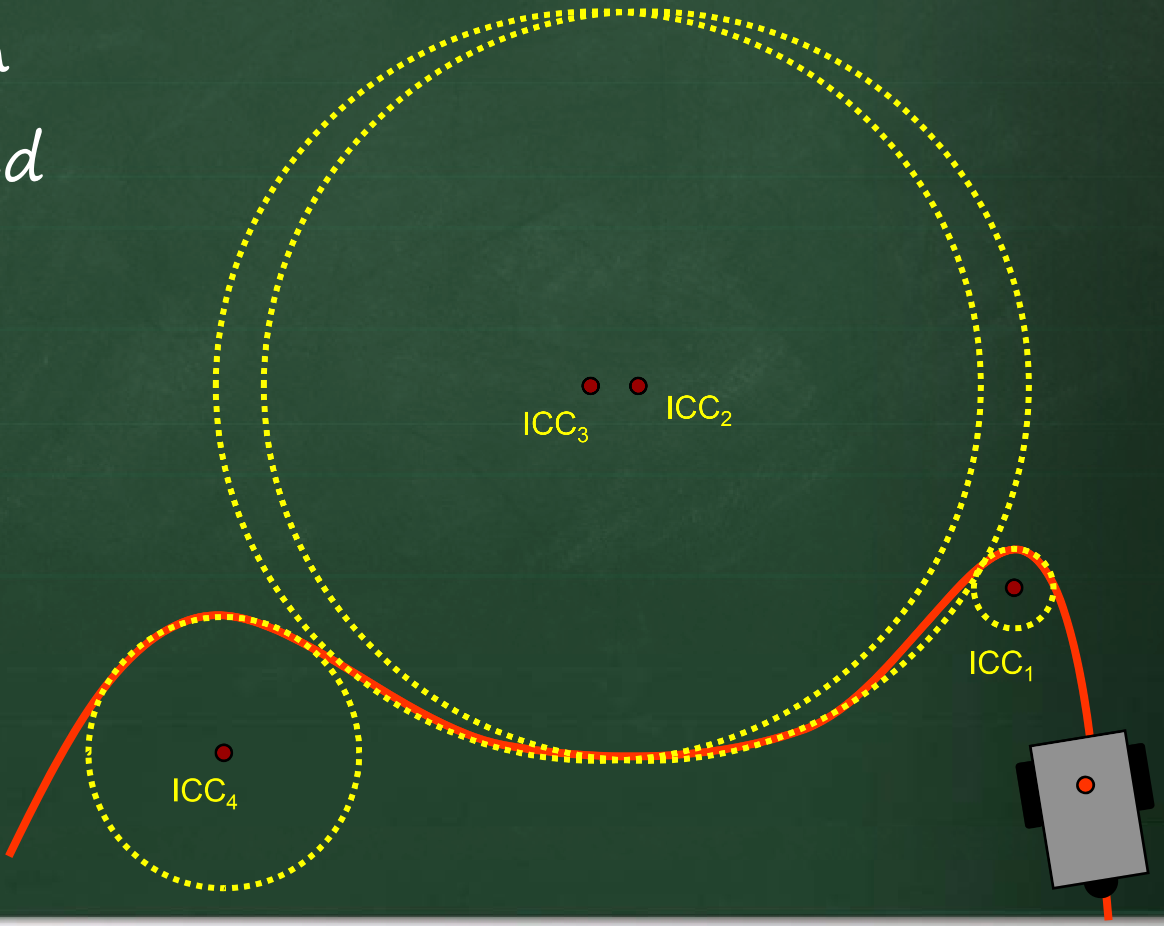
- Thus
  - if $x_{t+\delta} \neq x_t$:

    $\delta = (x_{t+\delta} - x_t) / (v_t \cos\theta_t)$ … we could have similarly fixed δ and solved for v_t
  - otherwise if $x_{t+\delta} = x_t$ then use:

    $\delta = (y_{t+\delta} - y_t)\ (v_t \sin\theta_t)$ … we could have similarly fixed δ and solved for v_t
- Hence set wheels at some fixed velocity v_t for amount of time δ to achieve desired travel distance.

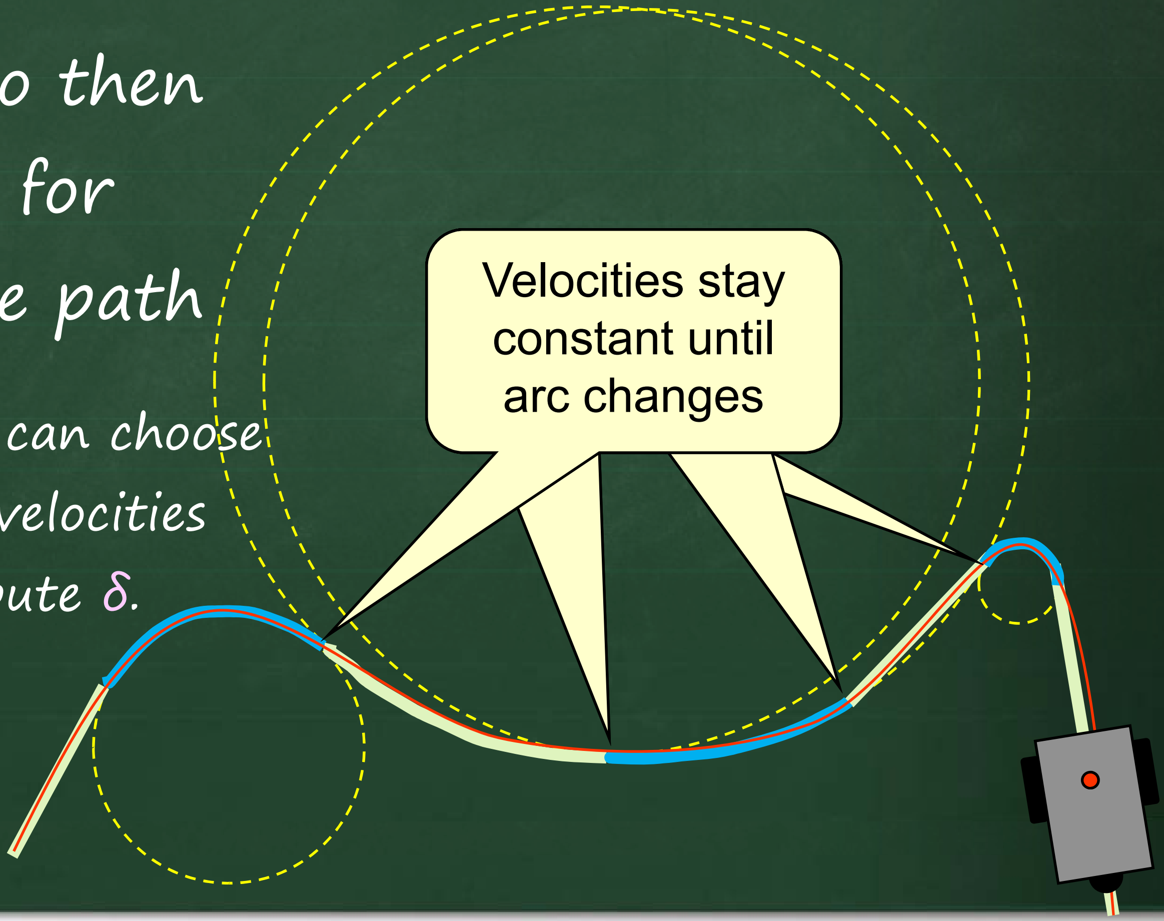
Inverse Kinematics

- Other options for inverse kinematics are to try and *approximate* a desired path with arcs based on manually computed ICC values:



Inverse Kinematics

- Result is a set of straight-line paths and ICC arc portions.
- Robot just needs to then compute velocities for each portion of the path
  - For each portion, you can choose to fix δ and compute velocities or fix speed and compute δ.



Kinematics Summary

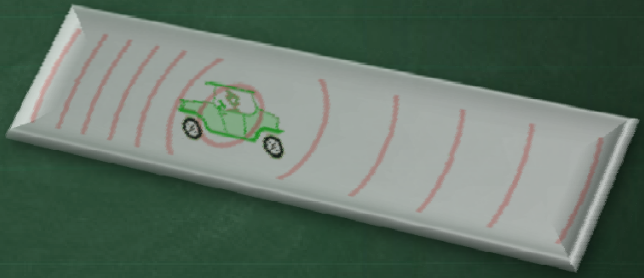
- Kinematic equations depend on robot's structure and wheel placement.
- For differential drive robots, we computed the equations necessary.
- Need to know the speed of each robot wheel at all times.

- Kinematics will have certain error associated with it due to the inability to know wheel velocities accurately.

Estimating Speed and Orientation Using Encoders

Determining Wheel Velocity

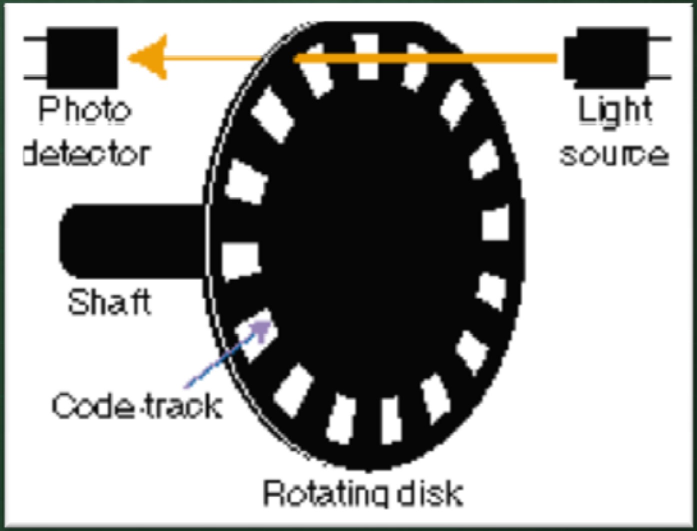
- Relative position estimation is extremely dependant on the measurement of the robot's velocity.
- Various sensors can be used to measure velocity:
  - Optical encoders on each wheel
  - Doppler sensors (usually ultrasonic)
  - Numerous industrial sensors



- For wheeled mobile robot applications, the most inexpensive and most popular are optical encoders.

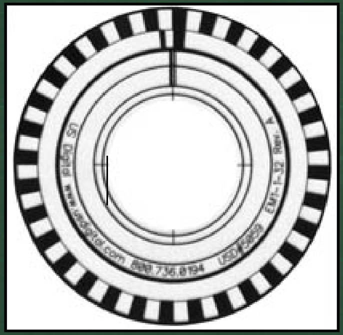
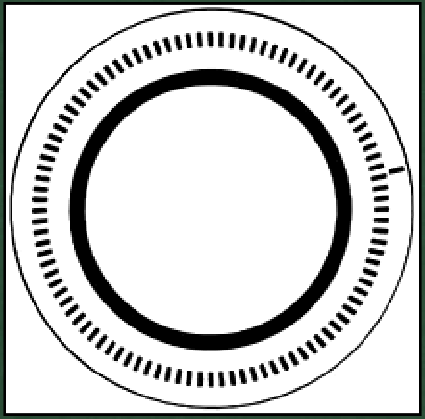
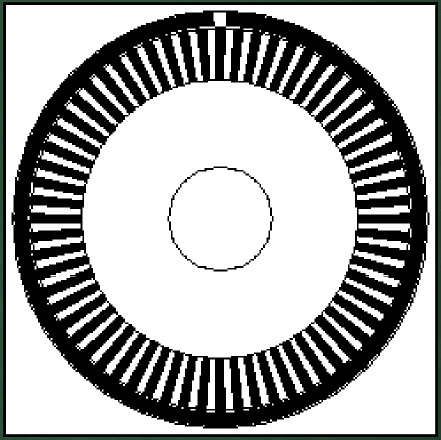
Optical Encoders

- *Optical encoders* are devices used to measure angular position and or velocity.
  - A focused beam of light aimed at a photodetector which is periodically interrupted by opaque or transparent pattern on a rotating disk attached to the shaft of the wheel.
- Two types:
  - *Incremental* encoders
  - *Absolute* encoders



Incremental Optical Encoders

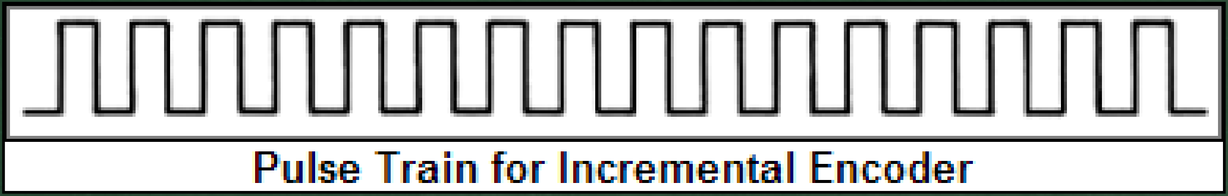
- Disk has evenly spaced slots around border which indicate accuracy:

- Can measure velocity and infer relative position.
- Two subtypes:
  - single-channel tachometer
  - phase-quadrature

Single-Channel Tachometer

- Spinning wheel produces *pulse train* square wave signal (i.e., 0 or 1 output)



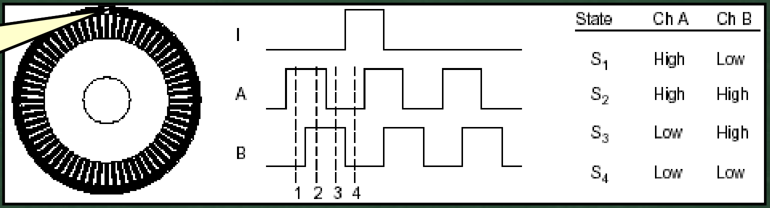
Makes "ticks" like placing a card in bicycle spokes.

- Advantages
  + Low cost, easy to interface for velocity measurement
- Disadvantages
  - Noise and stability problems at very low speeds
  - Cannot detect direction of rotation !!!

Phase-Quadrature Encoder

- 2nd channel added at 90° out of phase

Single slot used as 3rd channel for an index reference.

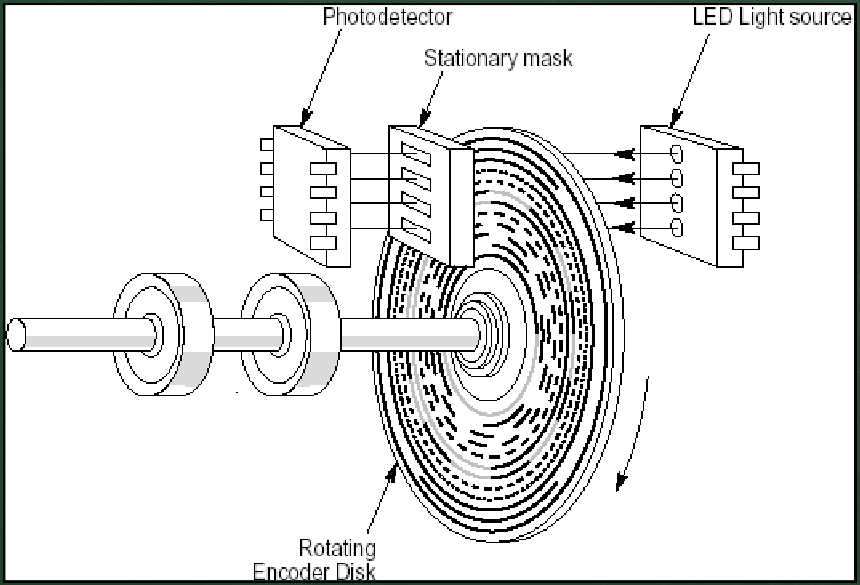


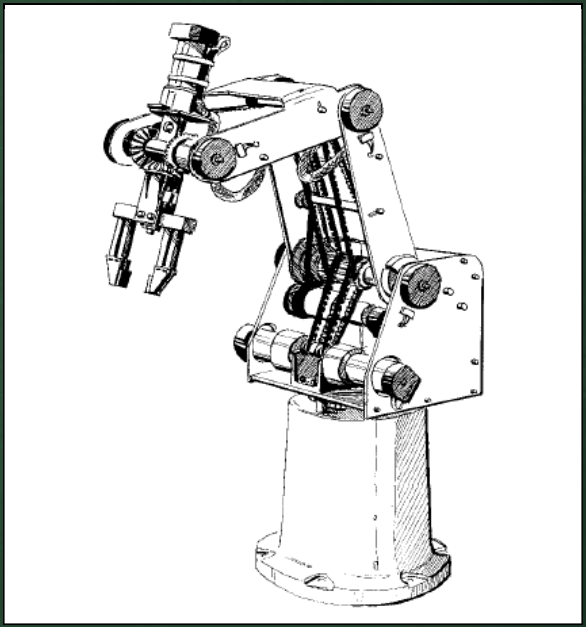
State	Ch A	Ch B
S_1	High	Low
S_2	High	High
S_3	Low	High
S_4	Low	Low

- Advantages
  + Can detect direction and angular position (relative to starting position)
  + 2 x more resolution than tachometer encoders
- Concerns
  - More difficult to interface
  - angular direction lost on power outage

Absolute Optical Encoders

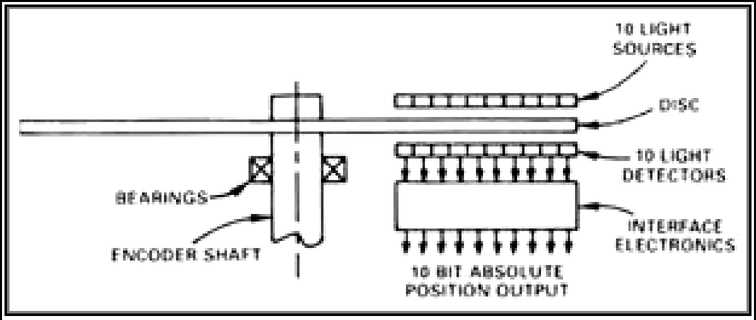
- Multiple light sources and photo detectors



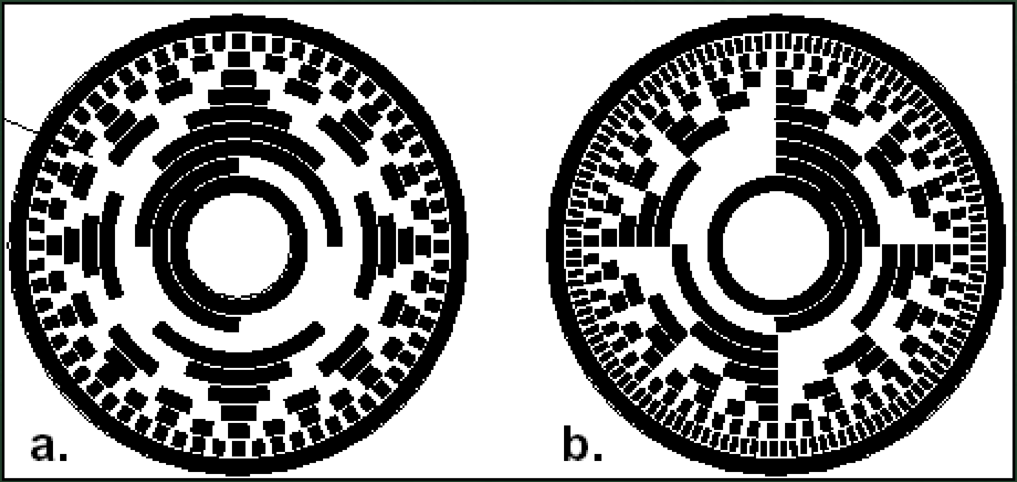


Absolute Optical Encoders

- Number of light/detector pairs indicates precision (e.g., 8-bit, 10-bit, 12-bit).
- Disks have a series of binary or gray-code patterns.

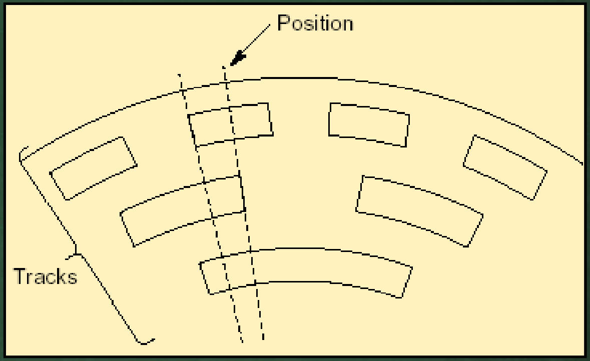


Absolute Optical Encoder Internal Construction



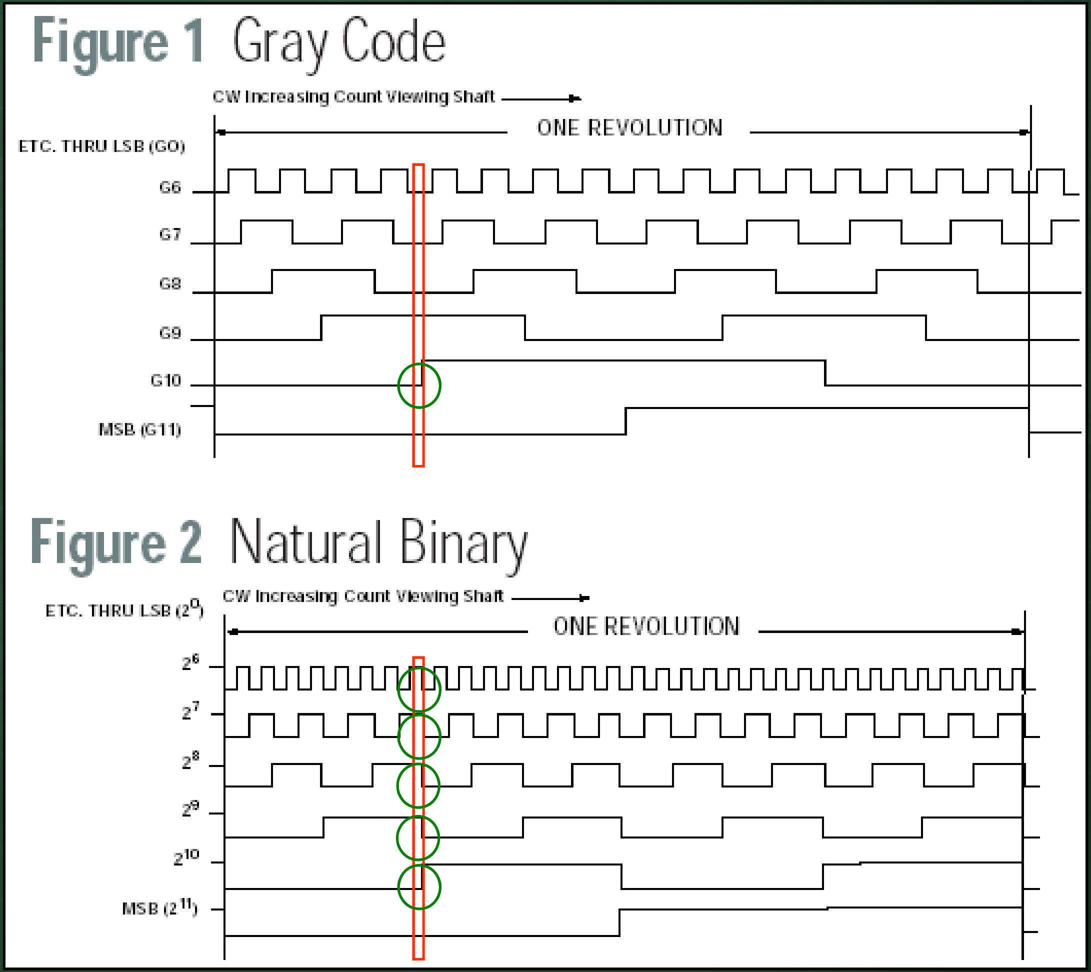
Absolute Optical Encoders

- The disk's angular position generates an x-bit binary reading.
- As the disk rotates, the readings from x tracks produce 2^x binary values.
- Measures actual absolute orientation of disk, independent of any previous readings.



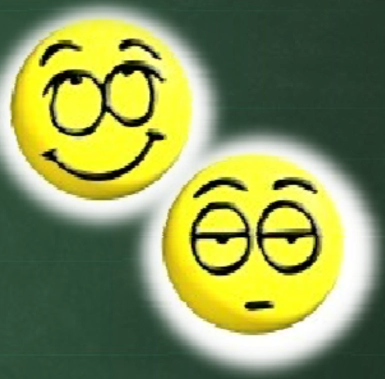
Absolute Optical Encoders

- *Gray code* is often *preferred* since at most 1 bit changes at once during rotation.
- *Reduces invalid readings* due to electrical ambiguity and mechanical component tolerances.



Comparing Optical Encoders

- Incremental encoders
  - + Easier to interface
  - + Lower cost
  - + Good for velocity and distance measurements
  - – angular position lost with noise or power loss
- Absolute encoders
  - + Always gives exact angular position
  - – More difficult to interface
  - – Higher cost
  - – Larger disk sizes required as bits increase
  - – Not meant for high speeds or distance measurement

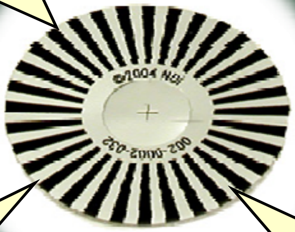


PropBot Odometry

PropBot Optical Encoders

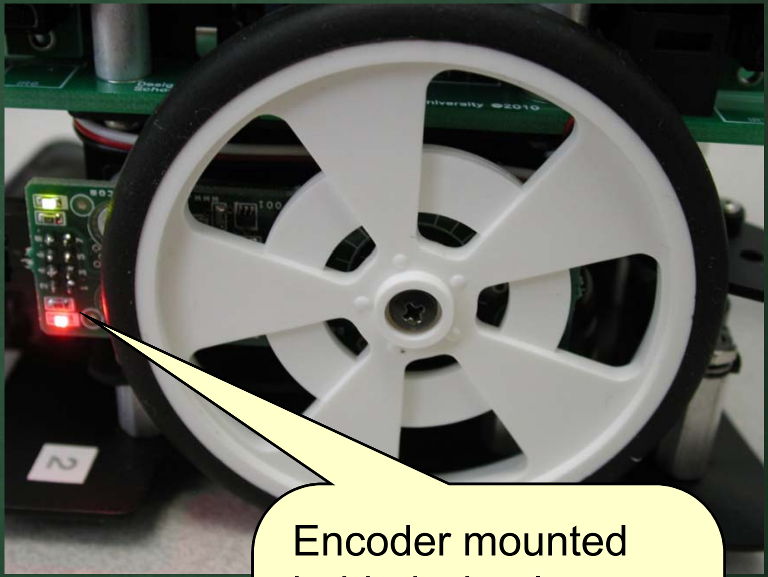
- Our PropBots come equipped with incremental optical quadrature encoders on each wheel
  - emits modulated IR light beam
  - IR reflection off wheel is detected

Disk contains 32 equally spaced striped wedges.

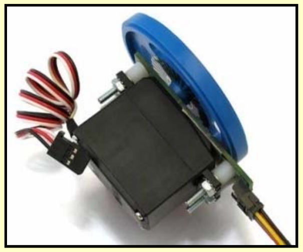


Quadrature reading gives 128 counts (or ticks) per revolution

Single channel reading gives 64 counts per revolution

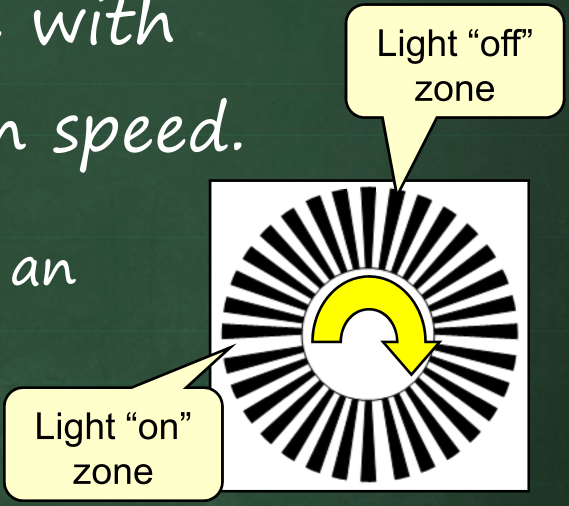


Encoder mounted behind wheel

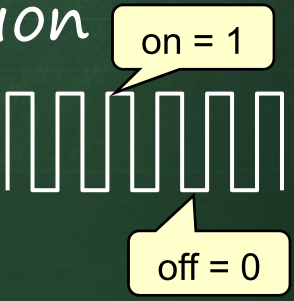


PropBot Optical Encoders

- Output of encoder is a square wave with frequency corresponding to rotation speed.
  - Each edge of the square wave will mark an increment of travel of approximately 0.1684cm (see slide 5-66).



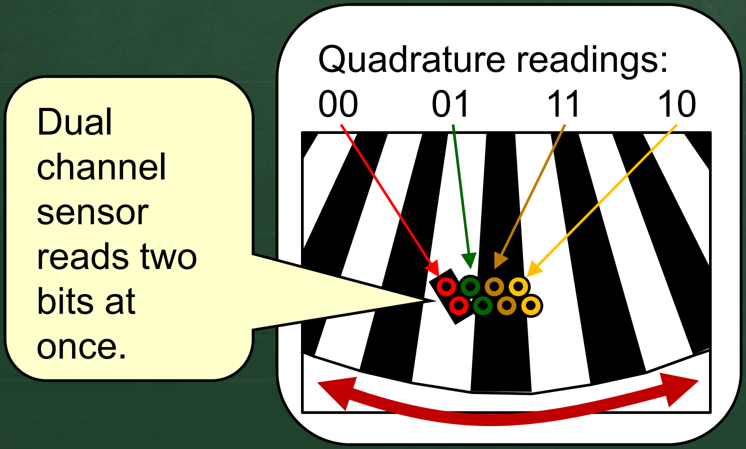
- Our code does not tell direction of rotation
  - our program operates servos as well hence direction is available

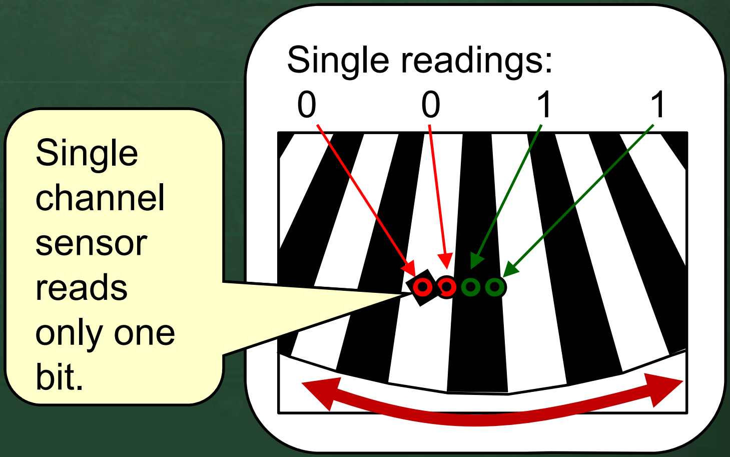


- Be aware that these encoders only provide a rough estimate of velocity and so measurement errors will accumulate quickly.

Measuring Distance

- Consider maintaining a PropBot's position.
  - As the wheel turns, the encoder provides either:
    - 128 two-bit signal pulses (i.e., dual channel)
    - 64 one-bit signal pulses (i.e., single channel)





Measuring Distance

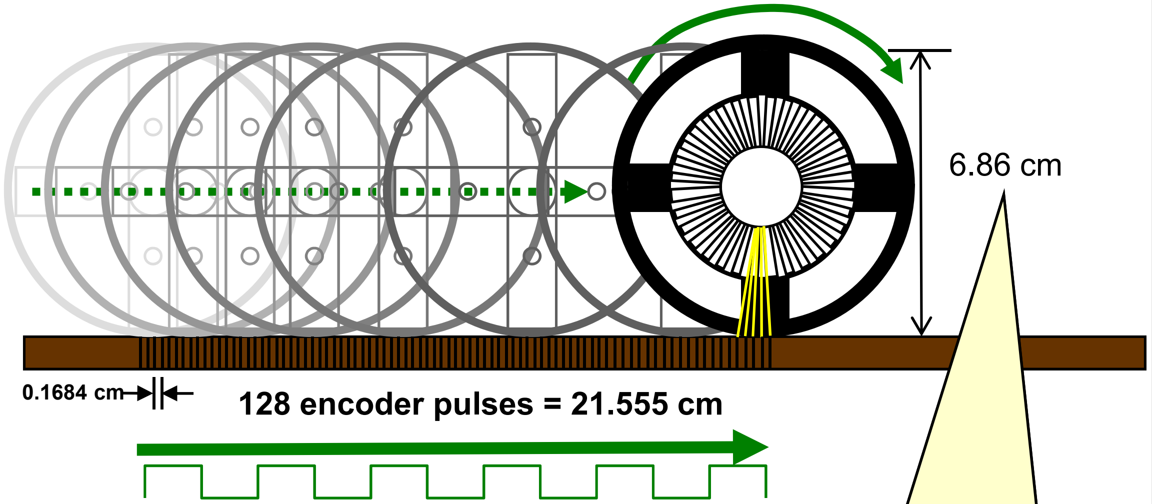
- If robot is moving straight ahead, simply count encoder pulses to determine its new location.

Distance traveled per pulse:

$$= \frac{(\text{Wheel Circumference})_{cm}}{128_{pulses}}$$

$= 6.86_{cm}\ \pi\ /\ 128$

$= 0.1684_{cm}$



Calculation assumes dual channel encoder. For single channel, the value is half as accurate: 0.3368cm travelled per pulse.

Wheel diameters in the lab actually vary ± 0.1cm, depending on the particular wheel.

Measuring Distance

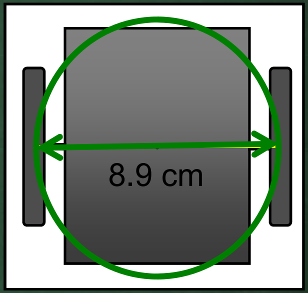
- Can easily update distance traveled upon pulse:

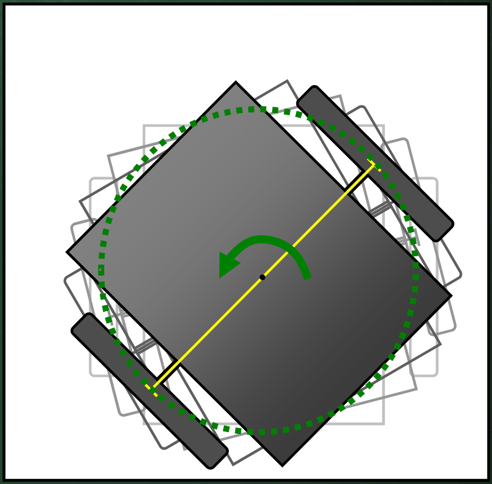
```
double   UNITS = 0.1684; // cm
double   distanceTraveled = 0.0;
boolean  previousLeft, currentLeft;

while (distanceTraveled < DESIRED_DISTANCE) {
  currentLeft = leftEncoder.detectsHole();
  if (currentLeft != previousLeft) {
    previousLeft = currentLeft;
    distanceTraveled += UNITS;
  }
}
```

- Distance will have error of up to 0.1684_{cm}.
  - For short distances, this can be serious
  - Error will accumulate and can cause problems.

Measuring Spin Angle

- Can easily measure angle changes when turning on the spot (i.e., $v^l = -v^r$)
  - Turning is centered around a circle with diameter of 8.9_{cm} and circumference of $\pi * 8.9_{cm} = 27.96_{cm}$
  - Since we know each encoder pulse indicates a travel distance of 0.1684_{cm}, each pulse therefore results in turn of: $0.1684_{cm/pulse} / 27.96_{cm} = 0.006023_{\%/pulse}$
  - Each pulse indicates a turn of: $0.006023_{\%/pulse} * 360° = \underline{2.17°}_{/pulse}$





Measuring Spin Angle

- Of course, this means that our turns may have an error of up to nearly 2.17° !!
- So if we simply want to move the robot forward and turn on the spot, these calculations are sufficient to maintain the position of the robot but will have some degree of error.
- Now what about turning with wheels moving at different speeds ?

Measuring Turn Angle

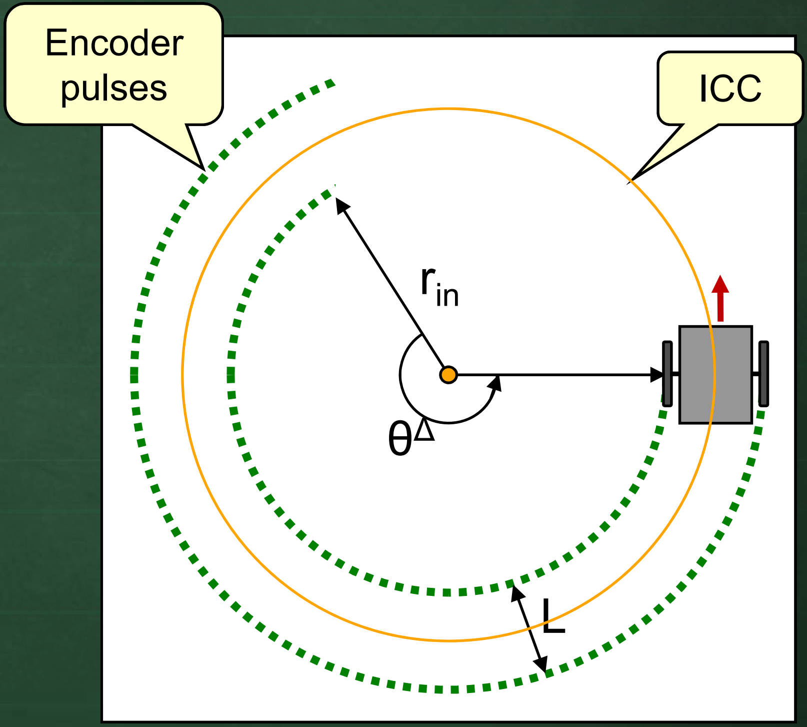
- As robot turns, each wheel traces out a circle with a different circumference:
  - Here $L = 8.9_{cm}$
  - Length of inner arc (left wheel):

    $D_{in} = r_{in} * \theta^{\Delta}$
  - Length of outer arc (right wheel):

    $D_{out} = (r_{in}+L) * \theta^{\Delta} = r_{in} * \theta^{\Delta} + L * \theta^{\Delta}$

    $= D_{in} + L * \theta^{\Delta}$
  - Hence

    $\theta^{\Delta} = (D_{out} - D_{in}) / L$



- D_{out} and D_{in} can be specified in terms of the encoder pulses p^r and p^l of the right and left wheels.

Measuring Turn Angle

- Hence, letting $p^{\Delta} = p^{r} - p^{l}$, the turn angle is:

$$\theta^{\Delta} = p^{\Delta}_{pulse} * 0.1684_{cm/pulse} / 8.9_{cm}$$
$$= 0.01892 p^{\Delta}_{rad}$$
$$= 1.0841\ p^{\Delta\circ}$$

We multiplied by $360°/ 2\pi_{rad}$

- What if $p^{l} = -p^{r}$ (as when spinning) ?
  - Then $\theta^{\Delta} = 1.0841 * 2p^{r\,\circ} = \underline{2.168}°\ p^{r}$

Same as we saw earlier when $p^{r} = 1$.

- So computing the turn angle is easy, but how do we determine the (x,y) position ?
- We can apply our forward kinematic equations by substituting encoder pulses for velocities.

PropBot Forward Kinematics

- Assume:
  - Robot begins at position (0,0) with angle $\theta = 0°$
  - Robot wheels set to unknown speeds V^r and V^l
    - we do not know the actual speeds in cm/sec
  - Robot counts number of left and right encoder pulses

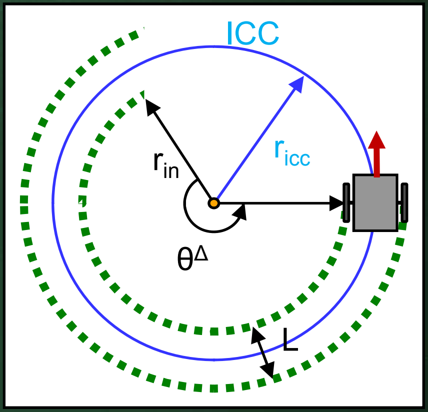
- Since, $D_{out} = (r_{in}+L)\ \theta^{\Delta}$ and $D_{in} = r_{in}\ \theta^{\Delta}$, we can compute radius r_{icc} of ICC as follows:

  $\theta^{\Delta} = D_{out} / (r_{in}+L)$ and $\theta^{\Delta} = D_{in} / r_{in}$

  Setting these equal yields, $D_{out} / (r_{in}+L) = D_{in} / r_{in}$

  and so, $r_{in} = LD_{in} / (D_{out} - D_{in})$

  $r_{icc} = r_{in} + L/2 = L(D_{in} / (D_{out} - D_{in}) + ½)$



PropBot Forward Kinematics

- We can write r_{icc} in terms of p^r and p^l as follows:

$$r_{icc} = L * (D_{in} / (D_{out} - D_{in}) + ½)$$
$$= 8.9_{cm} * (p^l*0.1684_{cm/pulse} / ((p^r - p^l)*0.1684_{cm/pulse})) + ½)$$
$$= 8.9_{cm} * (p^l / (p^r - p^l)) + 4.45_{cm}$$

- What about the angular velocity component, ω ?
  - We do not have velocities, nor a time component !!

- In fact, our computed value of $\theta^{\Delta} = 0.01892p^{\Delta}{}_{rad}$ is equivalent to the $\omega\delta$ component in our kinematic equations.

- t will represent the time when we start counting encoder pulses and $t+\delta$ when we stop counting.

PropBot Forward Kinematics

- So here are the forward kinematics for our PropBot:

$$x_{t+\delta} = r_{icc}\cos\theta^{\Delta}\sin\theta_t + r_{icc}\cos\theta_t\sin\theta^{\Delta} + x_t - r_{icc}\sin\theta_t$$

$$= x_t + r_{icc}\,(\cos\theta^{\Delta}\sin\theta_t + \cos\theta_t\sin\theta^{\Delta} - \sin\theta_t)$$

$$= x_t + r_{icc}\,(\sin(\theta_t+\theta^{\Delta}) - \sin\theta_t)$$

sin(X+Y) = sinXcosY + cosXsinY

$$y_{t+\delta} = r_{icc}\sin\theta^{\Delta}\sin\theta_t - r_{icc}\cos\theta_t\cos\theta^{\Delta} + y_t + r_{icc}\cos\theta_t$$

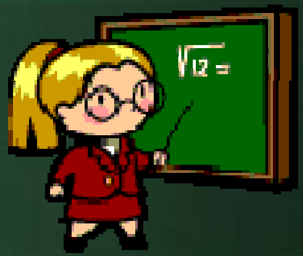
$$= y_t + r_{icc}\,(\sin\theta^{\Delta}\sin\theta_t - \cos\theta_t\cos\theta^{\Delta} + \cos\theta_t)$$

$$= y_t + r_{icc}\,(\cos\theta_t - \cos(\theta_t+\theta^{\Delta}))$$

cos(X+Y) = cosXcosY - sinXsinY

$$\theta_{t+\delta} = \theta_t + \theta^{\Delta}$$

The radius will be negative for the clockwise spin, but this is necessary for the above equations.



where, $r_{icc} = 8.9 * (p^l / (p^r - p^l)) + 4.45$

and $\theta^{\Delta} = [0.01892 * (p^r - p^l)]_{rad}$

Programming PropBot Kinematics

- We must not forget our special kinematic cases:
  - Straight motion: $p^r = p^l$
  - Spinning: $p^r = -p^l$
- In these cases, we apply our simpler kinematics as we did before
- For straight forward motion:

$$x_{t+\delta} = x_t + 0.1684 * p^r \cos(\theta_t)$$
$$y_{t+\delta} = y_t + 0.1684 * p^r \sin(\theta_t)$$
$$\theta_{t+\delta} = \theta_t$$

Programming PropBot Kinematics

- For spinning motion:

$$x_{t+\delta} = x_t$$

$$y_{t+\delta} = y_t$$

$$\theta_{t+\delta} = [\ \theta_t + \theta^{\Delta}\ ]$$

$$= \theta_t + [0.01892 * (p^r - p^l)]_{rad}$$

Robot Tracker Comparisons

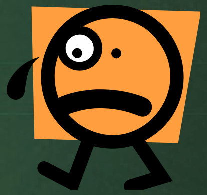
- When comparing our robot's estimated position with that of the RobotTracker, the units change.
  - Our formulas all use units of **cm**.
  - RobotTracker uses units of pixels (640x480 range).
- You will need to convert from cm to pixels or vice-versa. In the lab, 3 pixels = 1cm.
- Also, all our formulas assume that the coordinate system has (0, 0) as the bottom left corner. Our old RobotTracker version had the origin at the top left … so we had to invert the Y coordinate in that case.

PropBot Inverse Kinematics

- So now what about the inverse kinematics ?
- As before, we can do point-to-point travel by turning until facing the desired direction, and then moving forward.
- Is this accurate ? Consider forward motion …
  - We can only move in 0.1684_{cm} increments, so assuming "perfect travel conditions", we can be off by 0.1684_{cm}.
  - For short distances, this may be serious (e.g., 0.1684_{cm} on a 2_{cm} path is more than an 8% error)
  - If wheels speeds are slightly off then $p^r \neq p^l$ and our equations are off a little.

PropBot Inverse Kinematics

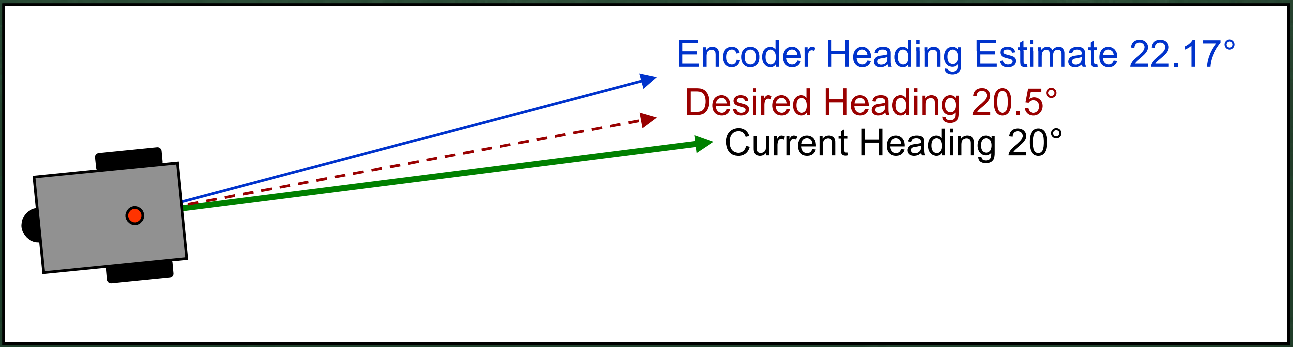
- As for the distance error of 0.1684_{cm}, there is not much we can do about it.



- If wheel speeds are slightly off, that is ok too.
  - Can simply keep count of left/right pulses and apply our forward kinematics to find out "how far off we were" once we arrive at our destination.
- If we want to **ensure** straight motion, we can monitor p^r and p^l as the robot travels along its "supposedly straight" line.
  - If ever ($p^r > p^l + 2$) or ($p^l > p^r + 2$) then we can make an adjustment, by leaving out one servo pulse to adjust.

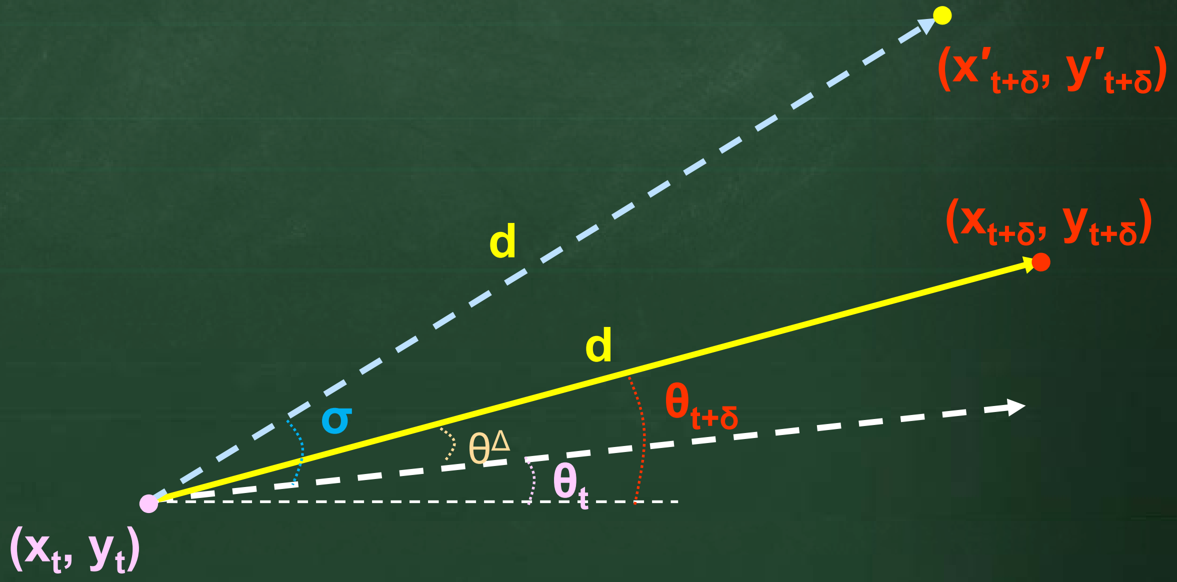
PropBot Inverse Kinematics

- Now what about the error obtained when spinning towards a heading ?
  - Recall each encoder pulse results in 2.17°/pulse
- So if we want to turn, say, 0.5° from our current heading, we cannot do this accurately !!



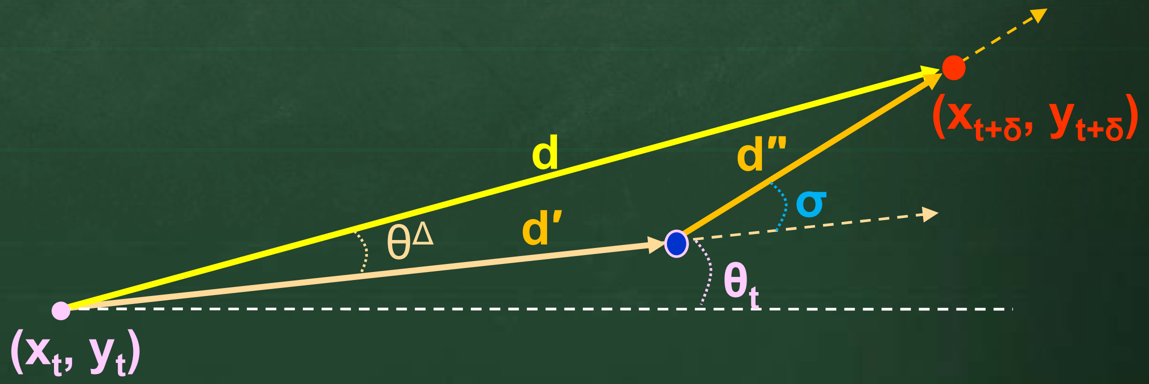
PropBot Path Improvement

- Can reduce error by applying a little "trick"
- Suppose robot needs to turn θ^{Δ}° & travel d_{cm} in a straight line from (x_t, y_t) to $(x_{t+\delta}, y_{t+\delta})$.
- In an attempt to turn to angle $\theta_{t+\delta} = \theta_t + \theta^{\Delta}$ the robot actually makes a turn of σ = 2.17° due to the coarseness of its encoders.
- If the robot were to travel distance d_{cm}, it would end up at position $(x'_{t+\delta}, y'_{t+\delta})$.



PropBot Path Improvement

- To help correct this problem, the robot can travel some distance d' in its current direction θ_t and then turn σ degrees towards $(x_{t+\delta}, y_{t+\delta})$ for distance d''.
- We just need to compute values d' and d''.
- Assume that $\sigma > \theta^{\Delta}$ otherwise we make some full σ turns until this condition is true.



PropBot Path Improvement

- Simple trigonometry tells us that:

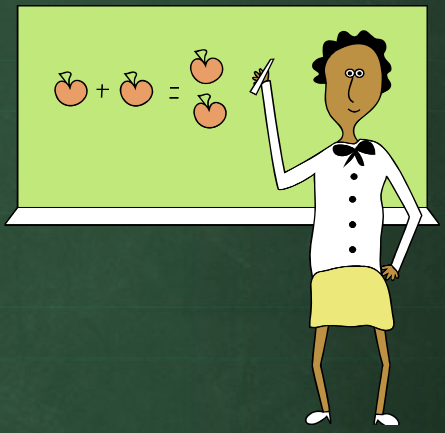
$$h = d'' \sin(\sigma - \theta^{\Delta}) \text{ and also } h = d' \sin(\theta^{\Delta})$$

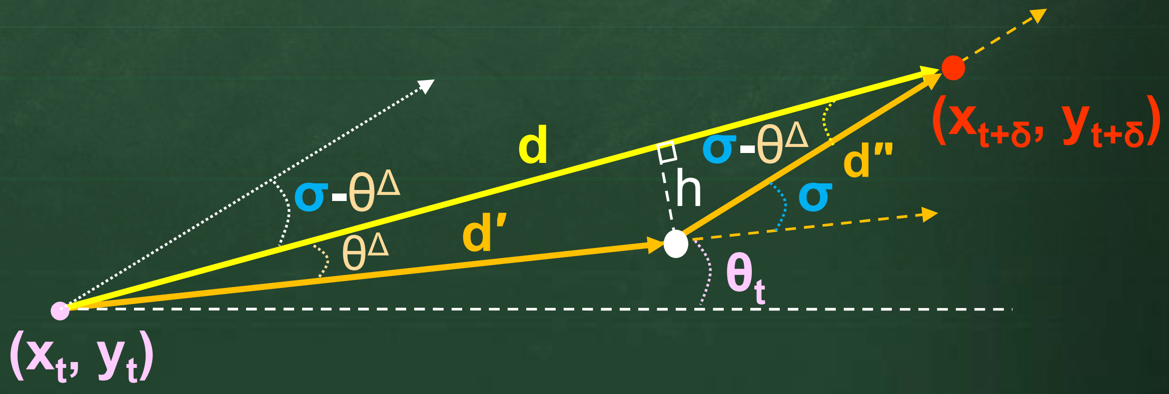
Setting these equal yields:

$$d'' \sin(\sigma - \theta^{\Delta}) = d' \sin(\theta^{\Delta})$$

and so

$$d'' = \frac{d' \sin(\theta^{\Delta})}{\sin(\sigma - \theta^{\Delta})}$$





PropBot Path Improvement

- Notice that here σ is the angle that is turned by one encoder pulse which is:

  $$2.17° = 0.037874_{rad}$$

- Note also that

  $$\sin(\sigma) = 0.037865_{rad} \approx \sigma.$$
  $$\cos(\sigma) = 0.99869_{rad} \approx 1.$$

- Similarly, for all $\theta^{\Delta} < \sigma$,

  look at **sin**(θ^{Δ}) and **cos**(θ^{Δ})

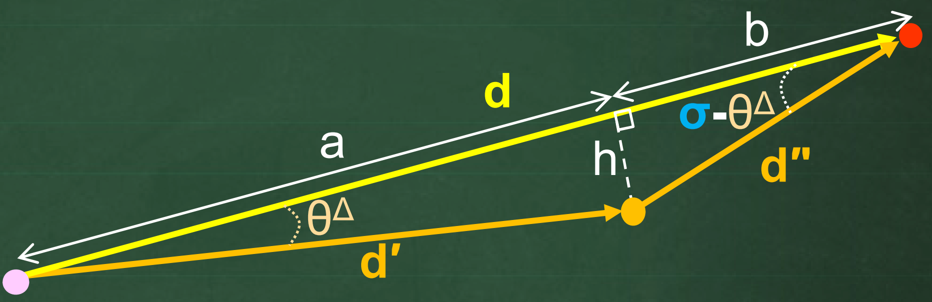
  Thus ...

  $$d'' = \frac{d' \sin(\theta^{\Delta})}{\sin(\sigma - \theta^{\Delta})} \approx \frac{d' \theta^{\Delta}}{(\sigma - \theta^{\Delta})}$$

θ^{Δ} (°)	θ^{Δ} (radians)	sin(θ^{Δ}) (radians)	cos(θ^{Δ}) (radians)
0	0.0	0.0	1.0
1	0.01745	0.01745	0.99985
2	0.03491	0.03490	0.99939
3	0.05236	0.05234	0.99863
4	0.06981	0.06976	0.99756
5	0.08727	0.08716	0.99619
6	0.10472	0.10453	0.99452
7	0.12217	0.12187	0.99255
8	0.13963	0.13917	0.99027
9	0.15708	0.15643	0.98769
10	0.17453	0.17365	0.98481
11	0.19199	0.19081	0.98163
12	0.20944	0.20791	0.97815
13	0.22689	0.22495	0.97437

PropBot Path Improvement

- Also, $a = d' \cos(\theta^{\Delta})$
  and $b = d'' \cos(\sigma - \theta^{\Delta})$



- Since $\sigma > \theta^{\Delta}$ then
  $\cos(\sigma) \approx \cos(\theta^{\Delta}) \approx \cos(\sigma - \theta^{\Delta}) \approx 1$ and so $d' + d'' \approx d$.

- So we plug these two assumptions together:

$$d'' = d'\,\theta^{\Delta} / (\sigma - \theta^{\Delta}) \quad \text{and} \quad d = d' + d''$$

thus,

$$d'' = d'\,\theta^{\Delta} / (\sigma - \theta^{\Delta}) = (d - d'')\,\theta^{\Delta} / (\sigma - \theta^{\Delta}) = d\theta^{\Delta} / \sigma$$

$$d' = d - d'' = d - d\theta^{\Delta} / \sigma = d(\sigma - \theta^{\Delta}) / \sigma$$

PropBot Path Improvement

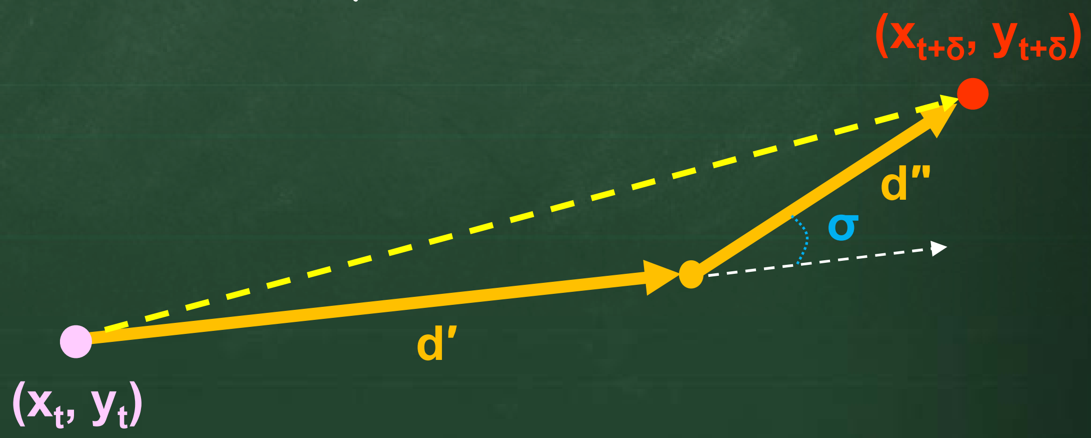
- For our single channel encoder we therefore compute:

$$d' = d(\sigma - \theta^{\Delta})/\sigma$$

$$= d(0.037874_{rad} - \theta^{\Delta}) / 0.037874_{rad}$$

$$= d(1 - 26.40334\ \theta^{\Delta})$$

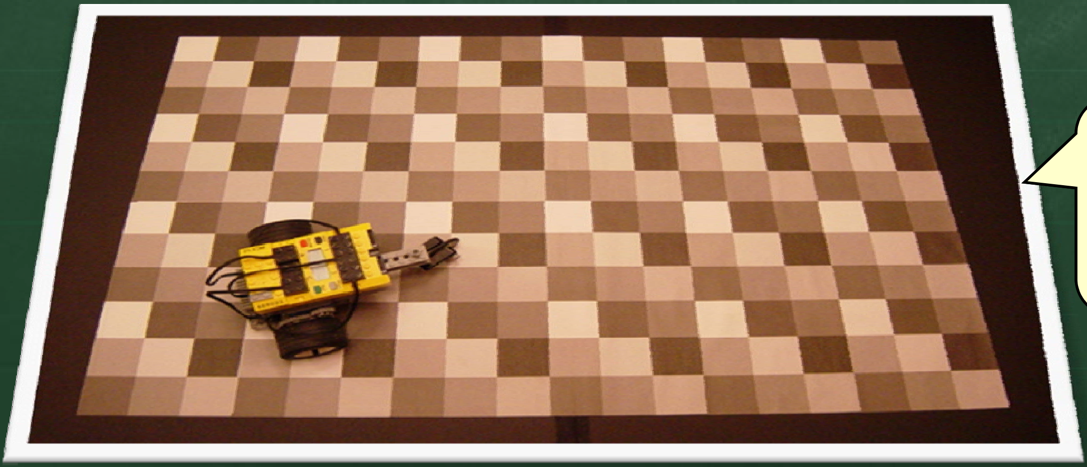
$$d'' = d - d'$$



Grid Tracking

Grid Tracking

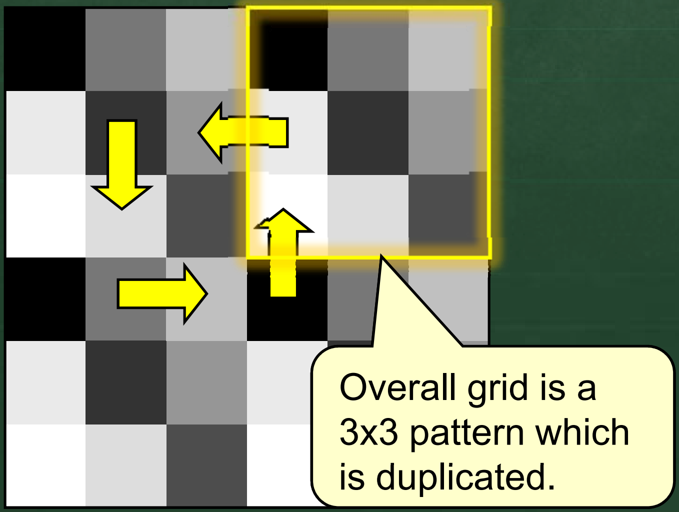
- Another strategy for position estimation is to do grid tracking.
  - place grid on floor with clearly identifiable grid cells
  - robot senses change from one cell to another

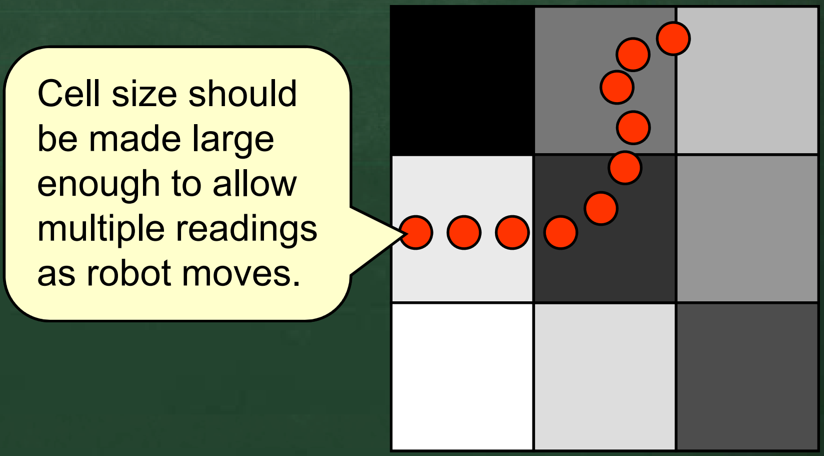


Results given here are from an honour's project by Dwayne Moore

Grid Design

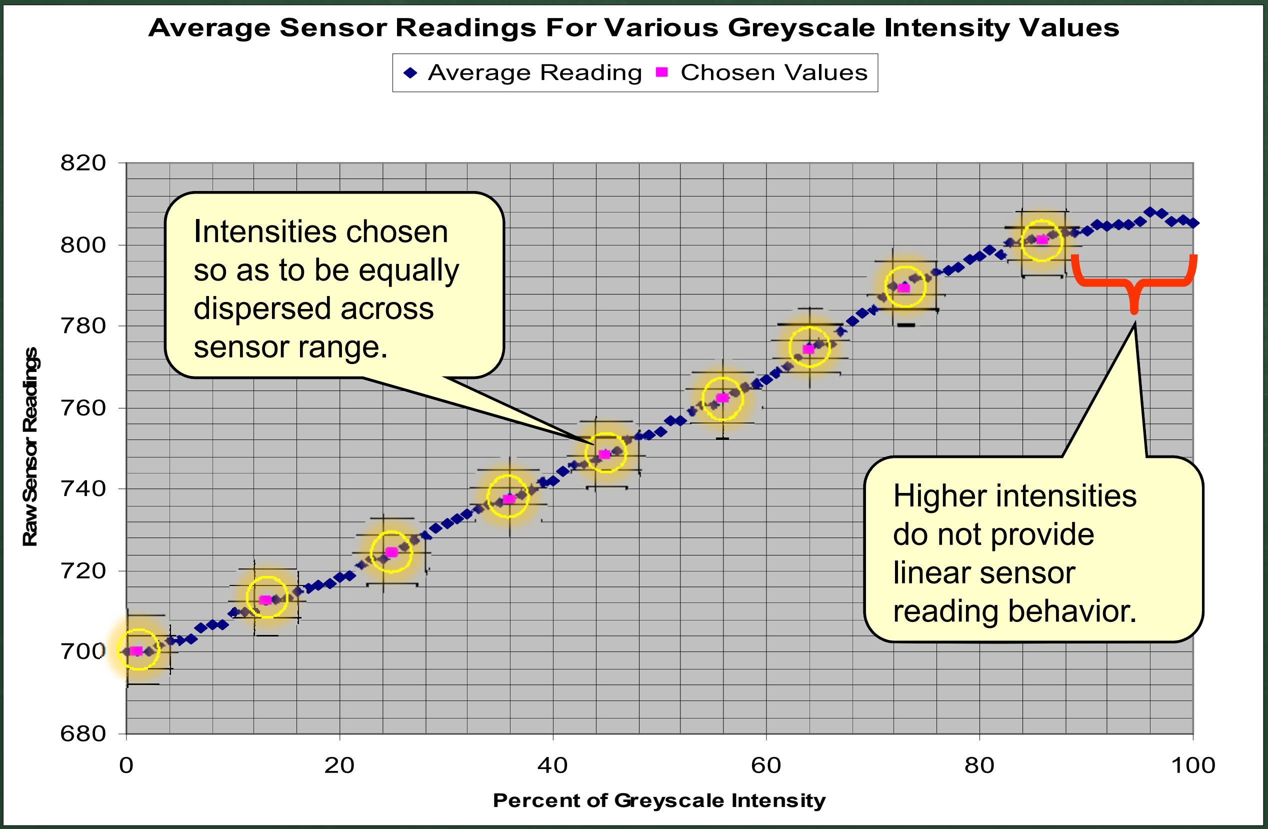
- Robot equipped with simple light intensity sensor
- Grid must be designed to distinguish changes from one cell to another:
  - need to maximize contrast between adjacent cells
  - grid cells must be larger when robot moves faster




Cell size should be made large enough to allow multiple readings as robot moves.

Grid Design

- Intensities of grid cells will depend on particular sensor (i.e., must be fine-tuned/calibrated)



Intensities chosen so as to be equally dispersed across sensor range.

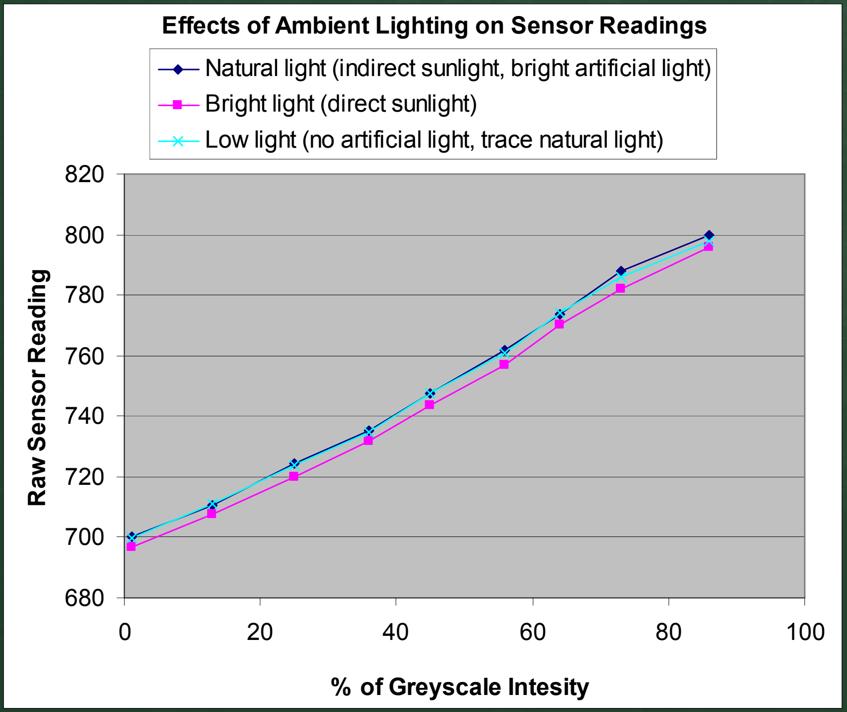
Higher intensities do not provide linear sensor reading behavior.

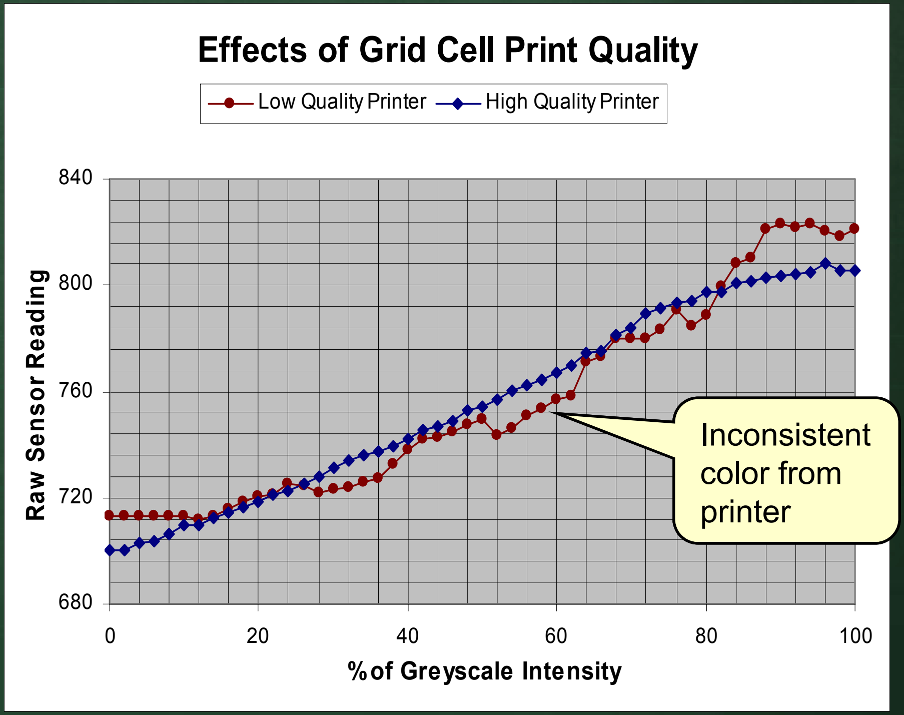
Greyscale Value	Sensor Reading
1	700
13	712
25	724
36	737
45	748
56	762
64	774
73	789
86	800

Cells are colored with these %'s of black.

Grid Design

- There are other factors to take into consideration when choosing cell colors and calibration values:

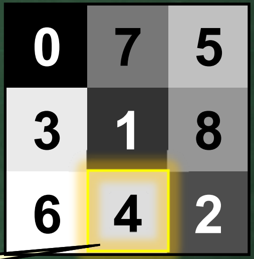




Identifying Grid Cells

- Robot determines the color of each grid cell through multiple readings, which may fluctuate.
  - lookup table can then be searched for closest value

700	712	724	737	748	762	774	789	800
0	1	2	3	4	5	6	7	8

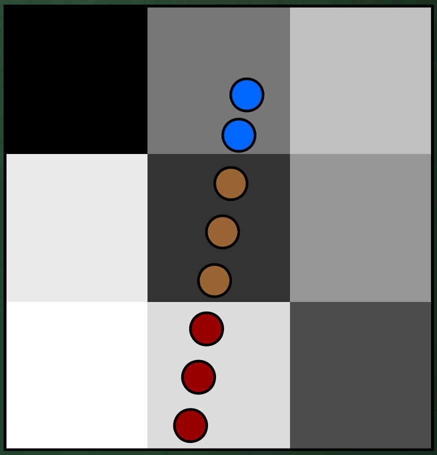


e.g., a reading of 752 would indicate this cell

  - As robot moves, readings will shift and the change should be noticeable:

e.g.. Readings may be as follows:

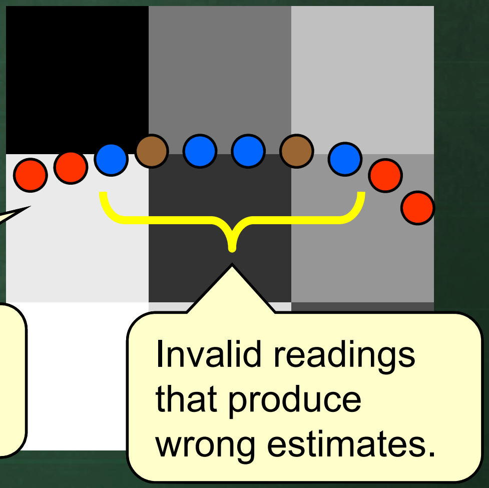
751, 751, 753, 716, 713, 711, 791, 788, …



Identifying Grid Cells

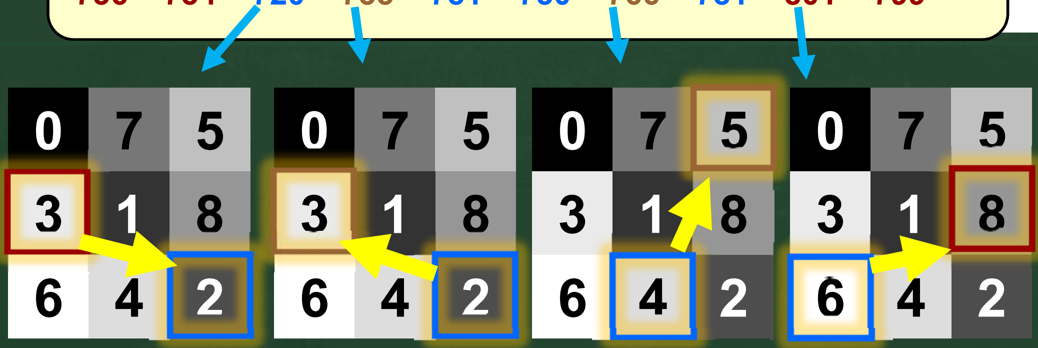
- Sensor readings will be weird and/or invalid along edges and corners.
- Need to try and detect:
  - take average of *many readings*
  - *discard highest/lowest* readings
  - *discard impossible* moves

700	712	724	737	748	762	774	789	800
0	1	2	3	4	5	6	7	8



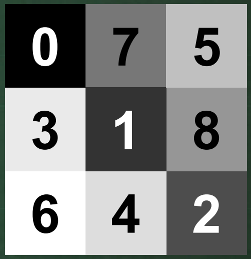
e.g.. Readings may be as follows:

736 – 734 – 720 – 735 – 751 – 750 – 766 – 781 – 801 – 799

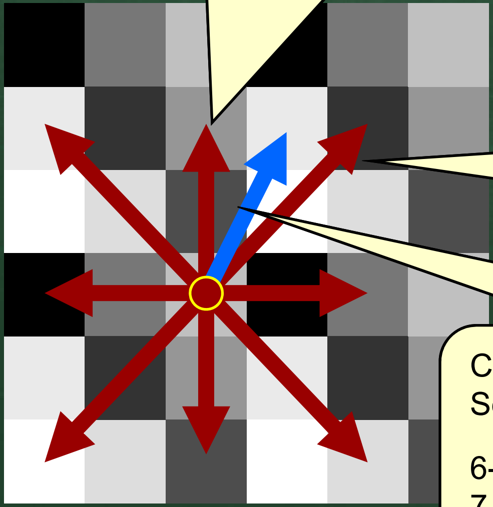


Identifying Direction

- Can identify direction by maintaining recent cells traversed.



0	7	5
3	1	8
6	4	2



Sequences that Identify **North** Direction:

6-3-0, 3-0-6, 0-6-3, 4-1-7, 1-7-4,
7-4-1, 2-8-5, 8-5-2, 5-2-8

Sequences that Identify **NorthEast** Direction:

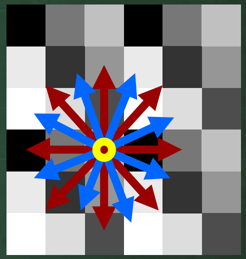
6-1-5, 1-5-6, 5-6-1, 3-7-2, 7-2-3,
2-3-7, 0-4-8, 4-8-0, 8-0-4

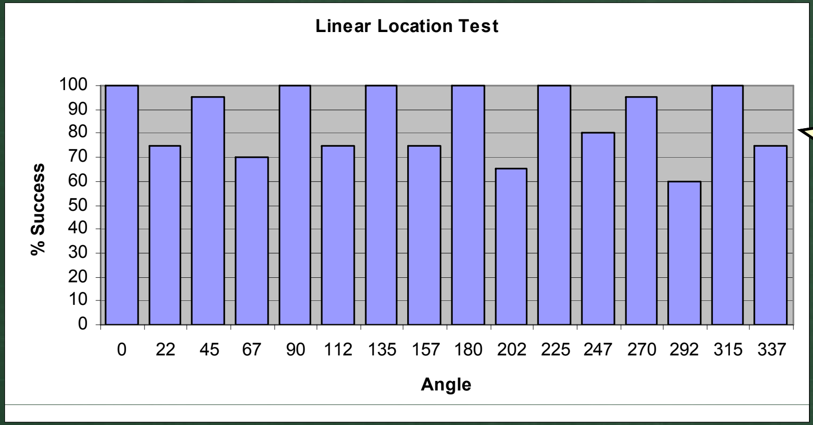
Can even identify 22.5 degree directions.
Sequences that Identify **North/NorthEast** Direction:

6-3-7, 3-0-4, 0-6-1, 4-1-5, 1-7-2,
7-4-8, 2-8-0, 8-5-6, 5-2-3

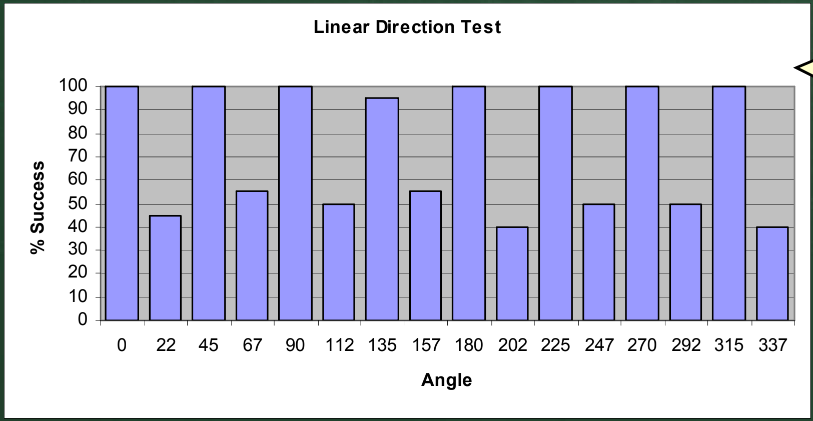
Results

- Here are results from some tests that sent the robot in the 16 different directions:





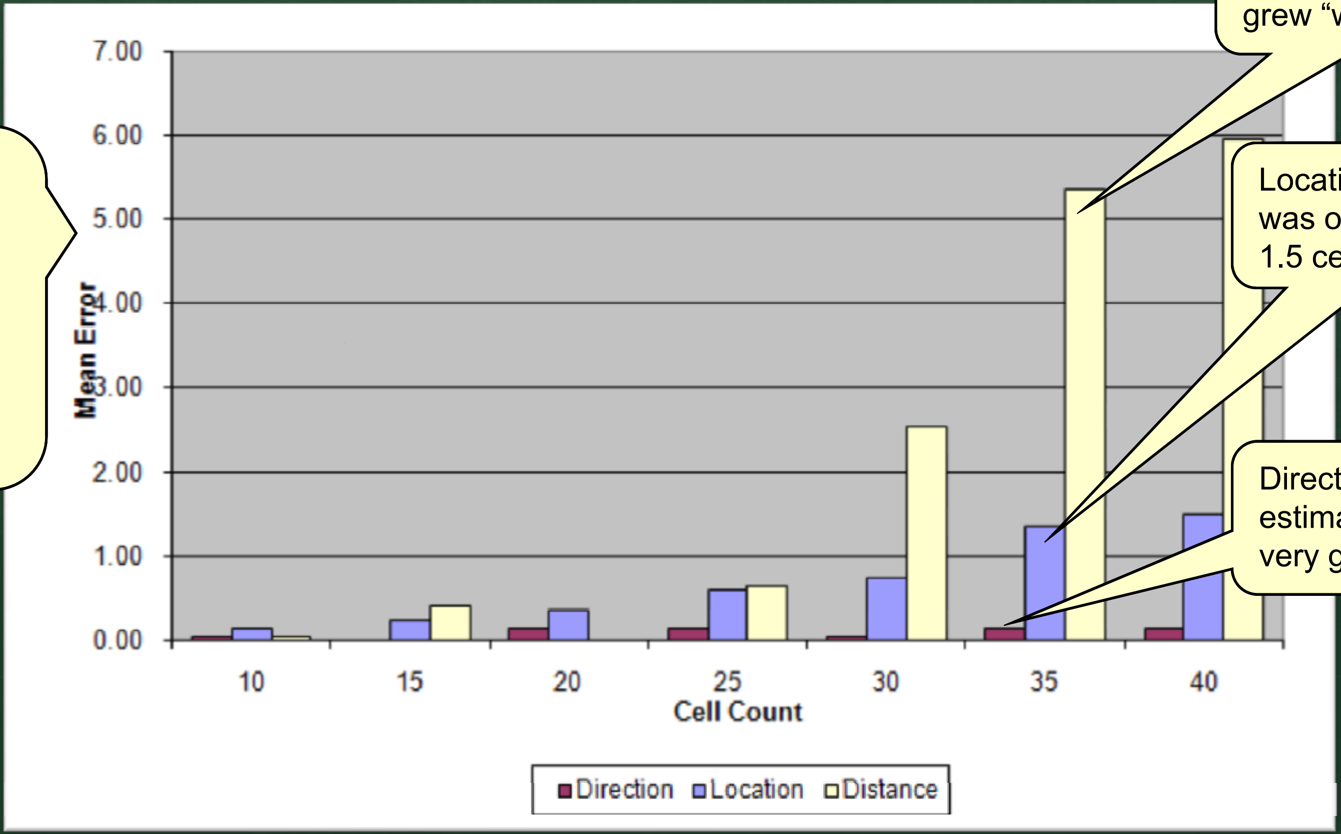
Robot was able to maintain location (i.e., grid coordinate) when traveling parallel, perpendicular and diagonal to grid cells, but had difficulty with finer angles.



Robot was able to maintain direction (i.e., grid angle) when traveling parallel, perpendicular and diagonal to grid cells, but had difficulty with finer angles.

Results

- Overall results from the project after about 5 feet of traveling through 40 cells:



Results could have been improved much more through further investigation and processing techniques.

Estimate of distance traveled grew "way off"

Location estimate was off by about 1.5 cells

Direction estimate was very good

Usefulness

- This approach has advantages:
  - + can re-confirm location after short distances and eliminate errors within 1 cell range
  - + simple to implement

- There are also disadvantages:
  - – cell size limits accuracy
  - – requires many sensor readings and large cells for truly reliable estimations
  - – requires modification of environment
  - – results depend on print quality and sensor calibration

Position Estimation Using Active Beacons

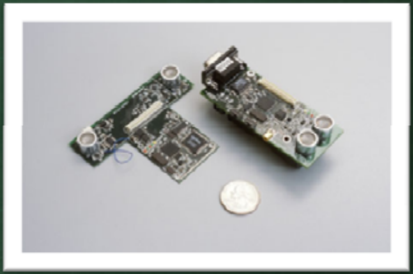
Active Beacons

- An *active beacon* is a stationary device (i.e., at fixed position in environment) that transmits and/or receives signals.

- Multiple beacons (2, 3 or more) must be installed to achieve proper position estimation.
- Commonly used for ships and airplanes.
- Robot estimates position and orientation by determining distance and angle to each of these beacons

Active Beacon Systems

- Active Beacon systems can produce:
  - + High accuracy in position estimation
  - + Provides quick sample rates
  - + Does not require high computational power
  - – Can be expensive to install and maintain
  - – Requires specialized environment



- There are many readily available systems:
  - Active Badge
  - Active Bat
  - RADAR
  - RICE project
  - E911
  - Cricket
  - MotionStar Magnetic Tracker
  - Easy Living
  - Smart Floor

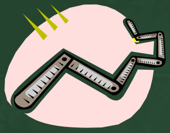
Active Beacon Systems

- Beacon systems based on triangulation
  - i.e., using geometric properties of triangles to compute position.

- Two types of triangulation techniques:
  - *lateration*
    - to determine robot <u>position</u> based on distance from beacons
    - 2D requires 3 non-colinear points (3D requires 4 non-coplanar)
  - *angulation*
    - to determine robot <u>position</u> and <u>angle</u> based on angle to beacons
    - 2D requires 2 angles and 1 known distance
    - 3D requires 1 additional azimuth measurement

Triangulation - Lateration

- 3 or more beacons emit a signal, robot obtains distance to each beacon:



Direct Measurement

- robot physically moves or sends probe etc..
- simple, but difficult to implement

Time-of-flight

- measure time it takes to travel to known point at specific velocity
- usually measure the difference in transmission and arrival time of an emitted signal (e.g., light, ultrasonic, radio).

Attenuation

- measure signal strength which decreases as distance from emission source increases.

Triangulation - Lateration

- Common time-of-flight emission types:
  - Ultrasonic
    - Sound moves at roughly $344_{m/sec}$ in 21°C air.

    e.g., Ultrasound pulse sent from A to B taking 14.5_{ms} indicates that $|\overline{AB}| \approx 5_m$.
  - Light
    - Light moves at roughly $299{,}792{,}458_{m/sec}$

    e.g., Light pulse sent from A to B taking 16.7_{ns} indicates that $|\overline{AB}| \approx 5_m$.

    Requires much higher clock resolution.
    (6 orders of magnitude !)
  - Radio
    - Radio moves fast like light at roughly $299{,}792{,}458_{m/sec}$
    - Typically emitted at various frequencies

Triangulation - Lateration

- Time-of-flight issues:
  - Pulses sent from point A will be reflected and so B may receive echoes/reflections indirectly:
    - these indirect signals are indistinguishable from direct.
    - can help alleviate problem by aggregating multiple receivers' measurements and observe reflective properties of the environment

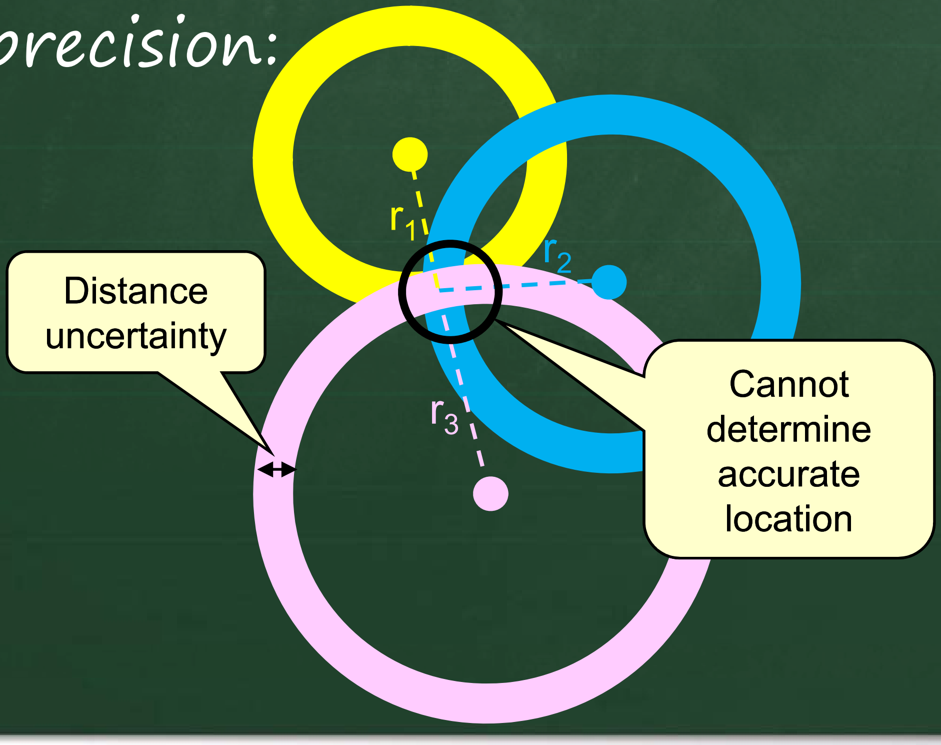
  - Time synchronization:
    - receiver at point B must know exact time that signal was emitted, otherwise timing, and hence distances, are wrong.
    - receiver must be synchronized precisely with sender.
    - GPS systems typically transmit the "time sent" within signal

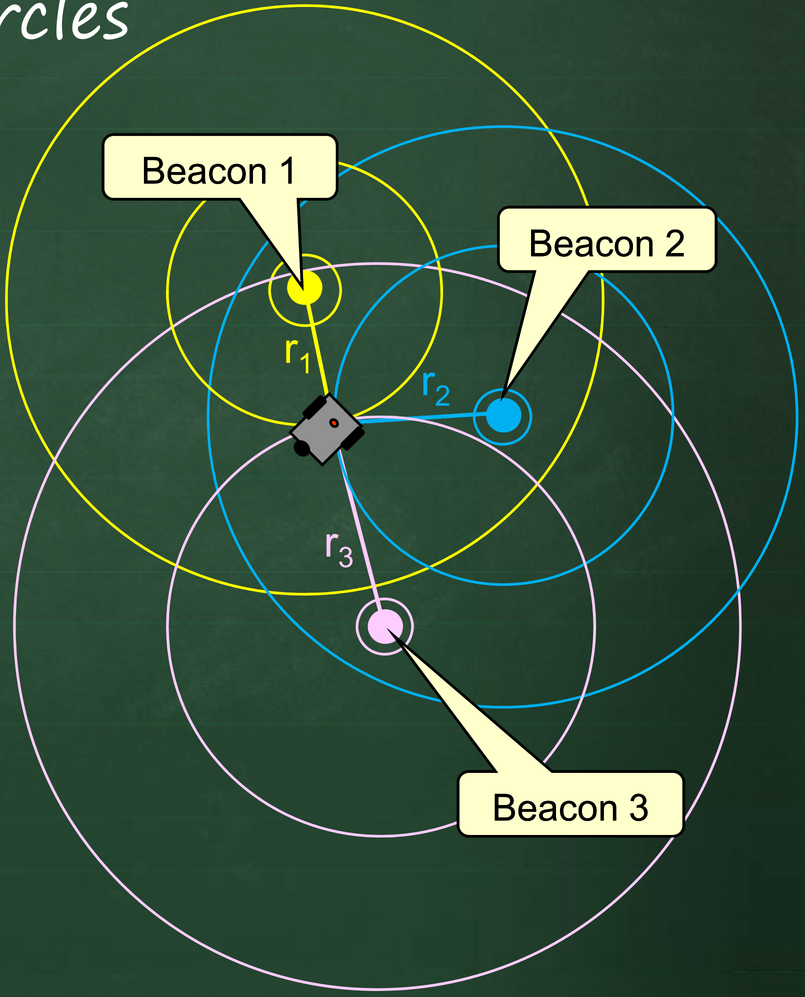
Triangulation - Lateration

- Attenuation issues:
  - Decrease in signal strength is not linear with increase in distance, making it difficult to model.
    - Free-space radio signal emitted from A will be attenuated (i.e., to become weaker in magnitude) by a factor of $1 / r^2$ where $r = |AB|$.
  - When there are many obstructions (e.g., indoor office space), inaccurate and imprecise distance measurements result due to propagation issues such as reflection, refraction and multi-path problems.

Triangulation - Lateration

- Location is intersection of 3 circles using distances as radii
- Accuracy depends on distance precision:





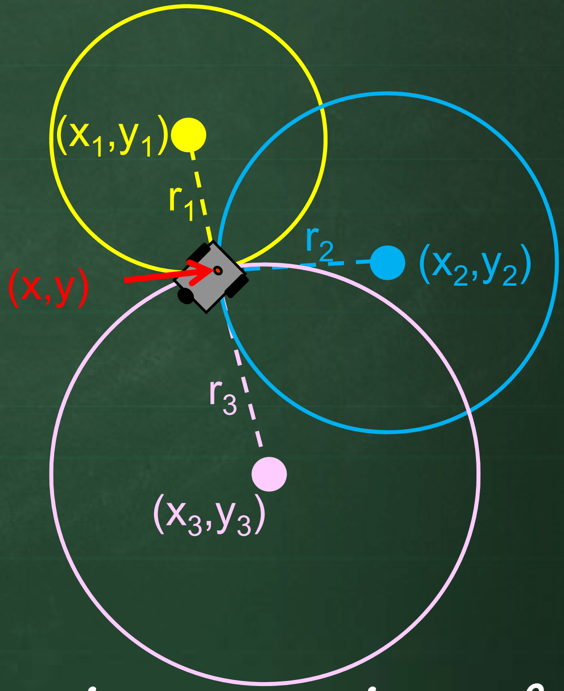
Triangulation - Lateration

- The equation of an origin-centered circle is:

  $x^2 + y^2 = r^2$.

- In our 3-beacon scenario we thus have the following equations:

$$(x - x_1)^2 + (y - y_1)^2 = r_1^2$$
$$(x - x_2)^2 + (y - y_2)^2 = r_2^2$$
$$(x - x_3)^2 + (y - y_3)^2 = r_3^2$$



- So the intersection point is simply the intersection of these 3 circles …

Triangulation – Lateration

- Setting the first two equations equal yields:

$(x - x_1)^2 + (y - y_1)^2 - r_1^2 = (x - x_2)^2 + (y - y_2)^2 - r_2^2$

$\rightarrow (x^2 - 2xx_1 + x_1^2) + (y^2 - 2yy_1 + y_1^2) - r_1^2 = (x^2 - 2xx_2 + x_2^2) + (y^2 - 2yy_2 + y_2^2) - r_2^2$

$\rightarrow -2xx_1 + x_1^2 - 2yy_1 + y_1^2 - r_1^2 = -2xx_2 + x_2^2 - 2yy_2 + y_2^2 - r_2^2$

$\rightarrow -2x(x_1 - x_2) + x_1^2 - x_2^2 - 2y(y_1 - y_2) + y_1^2 - y_2^2 - r_1^2 - r_2^2 = 0$

$\rightarrow -2x(x_1 - x_2) = 2y(y_1 - y_2) - x_1^2 + x_2^2 - y_1^2 + y_2^2 + r_1^2 + r_2^2$

$\rightarrow x = [2y(y_1 - y_2) - x_1^2 + x_2^2 - y_1^2 + y_2^2 + r_1^2 + r_2^2] / (-2(x_1 - x_2))$

$= y\,(y_2 - y_1) / (x_1 - x_2) + ½\,[x_1^2 - x_2^2 + y_1^2 - y_2^2 - r_1^2 - r_2^2] / (x_1 - x_2)$

Only works if ($\mathbf{x_1 \neq x_2}$) otherwise, solve for **y** instead of for **x** here

- We can find similar expressions for the intersection of circles 2 & 3 as well as 1 & 3:

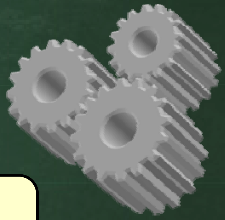
Be careful here too!

$x = y\,(y_3 - y_2)/(x_2 - x_3) + ½\,[x_2^2 - x_3^2 + y_2^2 - y_3^2 - r_2^2 - r_3^2]/(x_2 - x_3)$

$x = y\,(y_3 - y_1)/(x_1 - x_3) + ½\,[x_1^2 - x_3^2 + y_1^2 - y_3^2 - r_1^2 - r_3^2]/(x_1 - x_3)$

Triangulation - Lateration

- We can then combine any 2 of these together and then solve for y:



Believe it or not, much of this mess is actually constant !!

$$y\,(y_2 - y_1)/(x_1 - x_2) + ½\,[x_1^2 - x_2^2 + y_1^2 - y_2^2 - r_1^2 - r_2^2]/(x_1 - x_2)$$
$$= y\,(y_3 - y_2)/(x_2 - x_3) + ½\,[x_2^2 - x_3^2 + y_2^2 - y_3^2 - r_2^2 - r_3^2]/(x_2 - x_3)$$

$$\rightarrow y\,[(y_2 - y_1)/(x_1 - x_2) - (y_3 - y_2)/(x_2 - x_3)] = ½\,[x_2^2 - x_3^2 + y_2^2 - y_3^2 - r_2^2 - r_3^2]/(x_2 - x_3) - ½\,[x_1^2 - x_2^2 + y_1^2 - y_2^2 - r_1^2 - r_2^2]/(x_1 - x_2)$$

$$\rightarrow y = \{½\,[x_2^2 - x_3^2 + y_2^2 - y_3^2 - r_2^2 - r_3^2]/(x_2 - x_3) - ½\,[x_1^2 - x_2^2 + y_1^2 - y_2^2 - r_1^2 - r_2^2]/(x_1 - x_2)\}/[(y_2 - y_1)/(x_1 - x_2) - (y_3 - y_2)/(x_2 - x_3)]$$

- Plug resulting y value back into here to get x value

$$x = [2y(y_1 - y_2) - x_1^2 + x_2^2 - y_1^2 + y_2^2 + r_1^2 + r_2^2] / (-2(x_1 - x_2))$$

Triangulation - Lateration

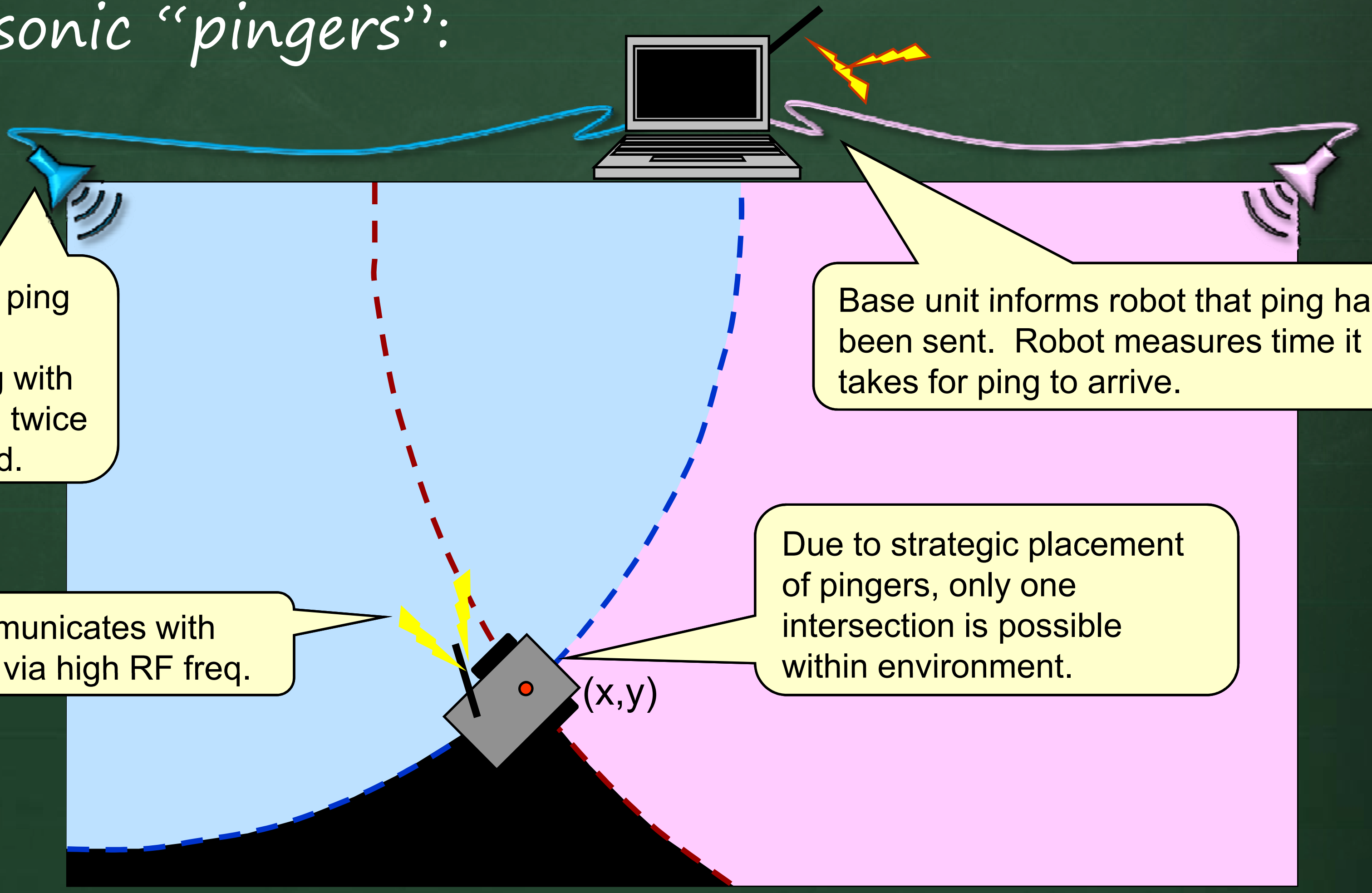
- Must be careful not to have a divide by zero in our equation when placing beacons.
- Since beacon positions are fixed, we end up with a simple equation based on fixed pre-computed constants as shown below:

$$(x,y) = C_0 + C_1 r_1^2 + C_2 r_2^2 + C_3 r_3^2$$

- There are other techniques for determining (x,y):
  - Geometric Triangulation
  - Newton-Raphson method (a root-finding algorithm)

Triangulation - Lateration

- Example of a typical system: **ISRobotics** uses two ultrasonic "pingers":



Ultrasonic ping emitted alternating with 2nd pinger, twice per second.

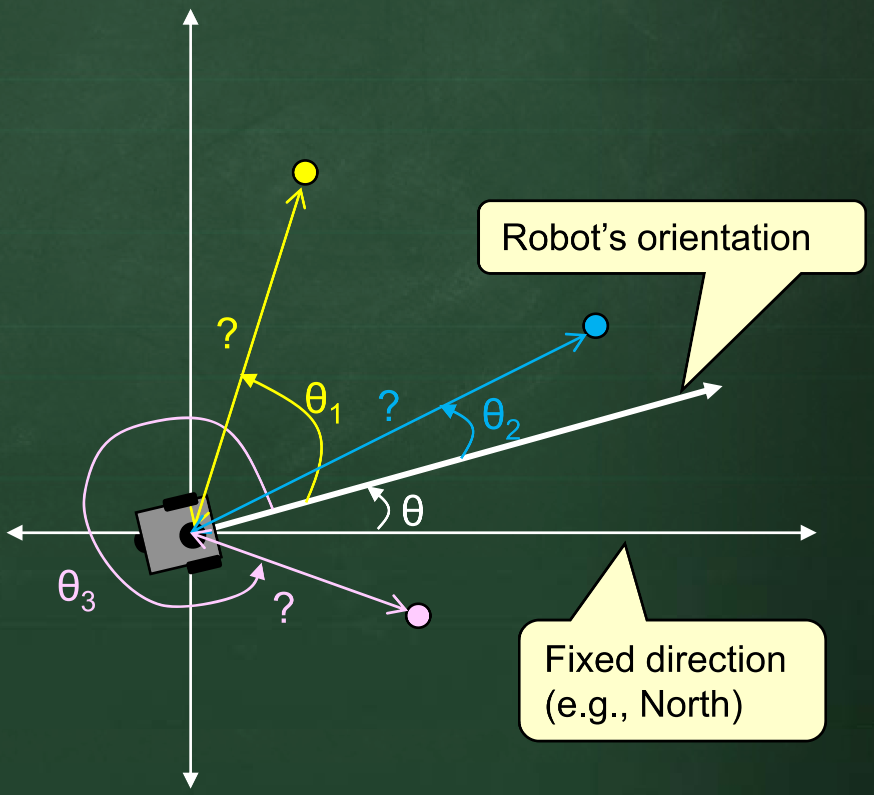
Base unit informs robot that ping has been sent. Robot measures time it takes for ping to arrive.

Communicates with base via high RF freq.

Due to strategic placement of pingers, only one intersection is possible within environment.

Triangulation – Angulation

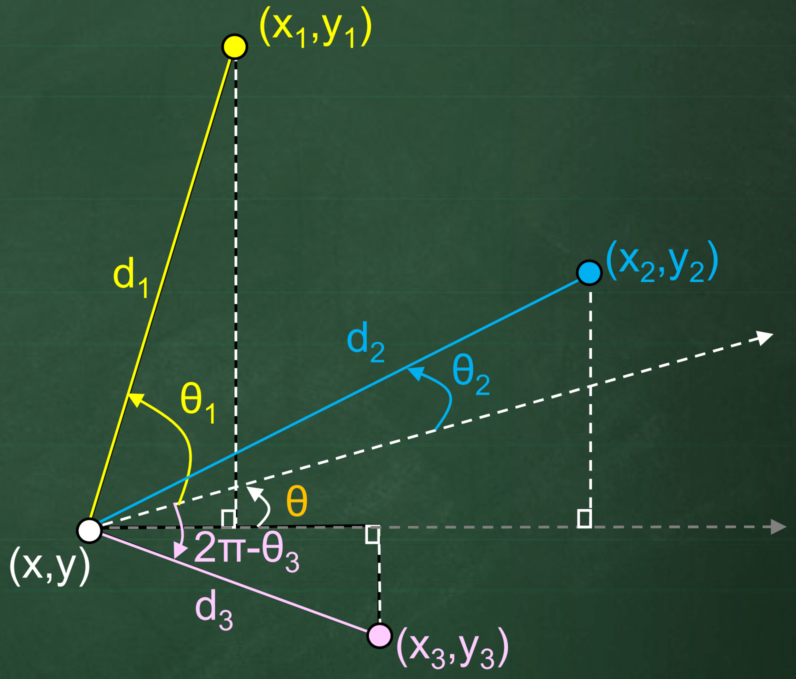
- *Angulation* is similar to lateration but makes use of angles to beacons as opposed to distances to them.
- Angles to beacons likely measured via rotation of receiver or transmitter on robot.
- Assumes all beacons are visible.



Triangulation – Angulation

- It is easy to see that:

$$d_1 \sin(\theta+\theta_1) = y_1 - y$$
$$d_1 \cos(\theta+\theta_1) = x_1 - x$$
$$d_2 \sin(\theta+\theta_2) = y_2 - y$$
$$d_2 \cos(\theta+\theta_2) = x_2 - x$$
$$d_3 \sin(\theta+\theta_3) = y_3 - y$$
$$d_3 \cos(\theta+\theta_3) = x_3 - x$$



- Setting these pairs equal:

$$(y_1 - y) \cos(\theta+\theta_1) = (x_1 - x) \sin(\theta+\theta_1)$$
$$(y_2 - y) \cos(\theta+\theta_2) = (x_2 - x) \sin(\theta+\theta_2)$$
$$(y_3 - y) \cos(\theta+\theta_3) = (x_3 - x) \sin(\theta+\theta_3)$$

- 3 equations…3 unknowns…sound familiar ?

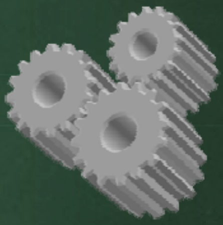
Triangulation - Angulation

- Rearranging:

$$(y_1 - y)\cos(\theta+\theta_1) = (x_1 - x)\sin(\theta+\theta_1)$$

$$\rightarrow y = y_1 + (x - x_1)\sin(\theta+\theta_1)/\cos(\theta+\theta_1)$$

$$\rightarrow y = y_1 + (x - x_1)\tan(\theta+\theta_1)$$



And similarly:

$$y = y_2 + (x - x_2)\tan(\theta+\theta_2)$$

$$y = y_3 + (x - x_3)\tan(\theta+\theta_3)$$

Setting any two of these equal yields:

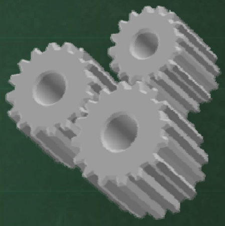
$$y_1 + (x - x_1)\tan(\theta+\theta_1) = y_2 + (x - x_2)\tan(\theta+\theta_2)$$

$$y_1 + (x - x_1)\tan(\theta+\theta_1) = y_3 + (x - x_3)\tan(\theta+\theta_3)$$

$$y_2 + (x - x_2)\tan(\theta+\theta_2) = y_3 + (x - x_3)\tan(\theta+\theta_3)$$

Triangulation - Angulation

- Now we have 2 equations and 2 unknowns.
- We can break them down to further obtain:



$$x\tan(\theta+\theta_1) - x\tan(\theta+\theta_2) = y_2 - y_1 + x_1\tan(\theta+\theta_1) - x_2\tan(\theta+\theta_2)$$

$$\rightarrow x = [y_2 - y_1 + x_1\tan(\theta+\theta_1) - x_2\tan(\theta+\theta_2)] / (\tan(\theta+\theta_1) - \tan(\theta+\theta_2))$$

And thus similarly these two can be obtained:

$$x = [y_3 - y_1 + x_1\tan(\theta+\theta_1) - x_3\tan(\theta+\theta_3)]/(\tan(\theta+\theta_1) - \tan(\theta+\theta_3))$$

$$x = [y_3 - y_2 + x_2\tan(\theta+\theta_2) - x_3\tan(\theta+\theta_3)]/(\tan(\theta+\theta_2) - \tan(\theta+\theta_3))$$

- Setting any pair of these together, we can eliminate x and have only one unknown.

Triangulation - Angulation

- Hence, for example:

$$\frac{y_2 - y_1 + x_1\tan(\theta+\theta_1) - x_2\tan(\theta+\theta_2)}{\tan(\theta+\theta_1) - \tan(\theta+\theta_2)} = \frac{y_3 - y_1 + x_1\tan(\theta+\theta_1) - x_3\tan(\theta+\theta_3)}{\tan(\theta+\theta_1) - \tan(\theta+\theta_3)}$$

- So we just need to solve for θ … not so easy… but it can be done.
- In many real situations, we already have value of θ
  - From a digital compass or
  - Estimated from odometry

Triangulation – Angulation

- If this is the situation, we can simply plug in θ into our equations for x and y (being careful to watch out for the division by zero of course).
- Typical *Geometric Triangulation* applications assume that the robot will be within the area defined by 3 or more beacons.
- In this case, we can try a different approach.

Triangulation – Angulation

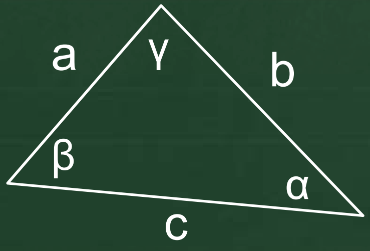
- We can take advantage of known locations of beacons and the distances

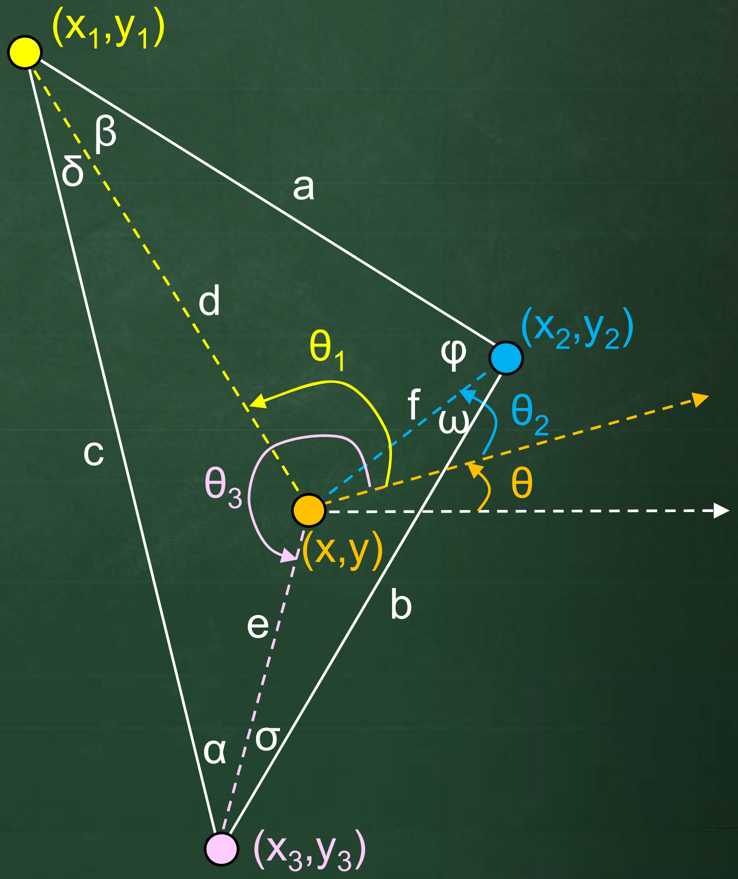
  & angles between them.

- Make use of formulae:

$$a^2 = b^2 + c^2 - 2bc\cos(\alpha)$$

$$\sin(\alpha)/a = \sin(\beta)/b = \sin(\gamma)/c$$





Triangulation – Angulation

- In this case we know many things:

$\sin(\theta_3 - \theta_1) / c = \sin(\alpha) / d$

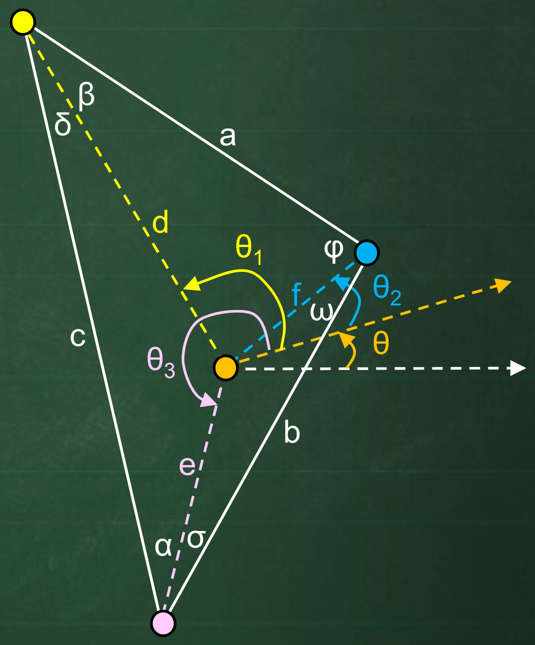
$\sin(\theta_1 - \theta_2) / a = \sin(\phi) / d$

$\sin(2\pi - (\theta_3 - \theta_2)) / b = \sin(\sigma) / f$

$\sin(\theta_1 - \theta_2) / a = \sin(\beta) / f$

$\sin(\theta_3 - \theta_1) / c = \sin(\delta) / e$

$\sin(2\pi - (\theta_3 - \theta_2)) / b = \sin(\omega) / e$



- And so,

$c \cdot \sin(\alpha) / \sin(\theta_3 - \theta_1) = d = a \cdot \sin(\phi) / \sin(\theta_1 - \theta_2)$

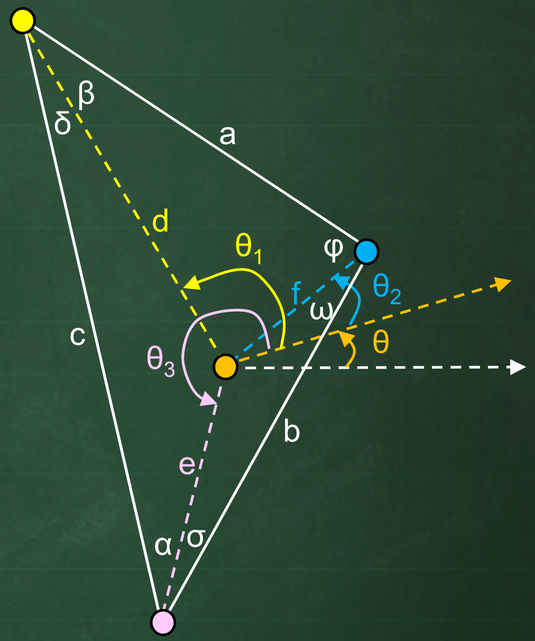
$b \cdot \sin(\sigma) / \sin(2\pi - (\theta_3 - \theta_2)) = f = a \cdot \sin(\beta) / \sin(\theta_1 - \theta_2)$

$c \cdot \sin(\delta) / \sin(\theta_3 - \theta_1) = e = b \cdot \sin(\omega) / \sin(2\pi - (\theta_3 - \theta_2))$

Triangulation – Angulation

- We also know that:

$$\theta_1 - \theta_2 + \phi + \beta = 180°$$
$$\theta_3 - \theta_1 + \delta + \alpha = 180°$$
$$2\pi - \theta_3 + \theta_2 + \omega + \sigma = 180°$$
$$\phi + \beta + \delta + \alpha + \omega + \sigma = 180°$$
$$\sin(\alpha + \sigma) / a = \sin(\delta + \beta) / b = \sin(\phi + \omega) / c$$



- Plugging the first three of these into our equations yields:

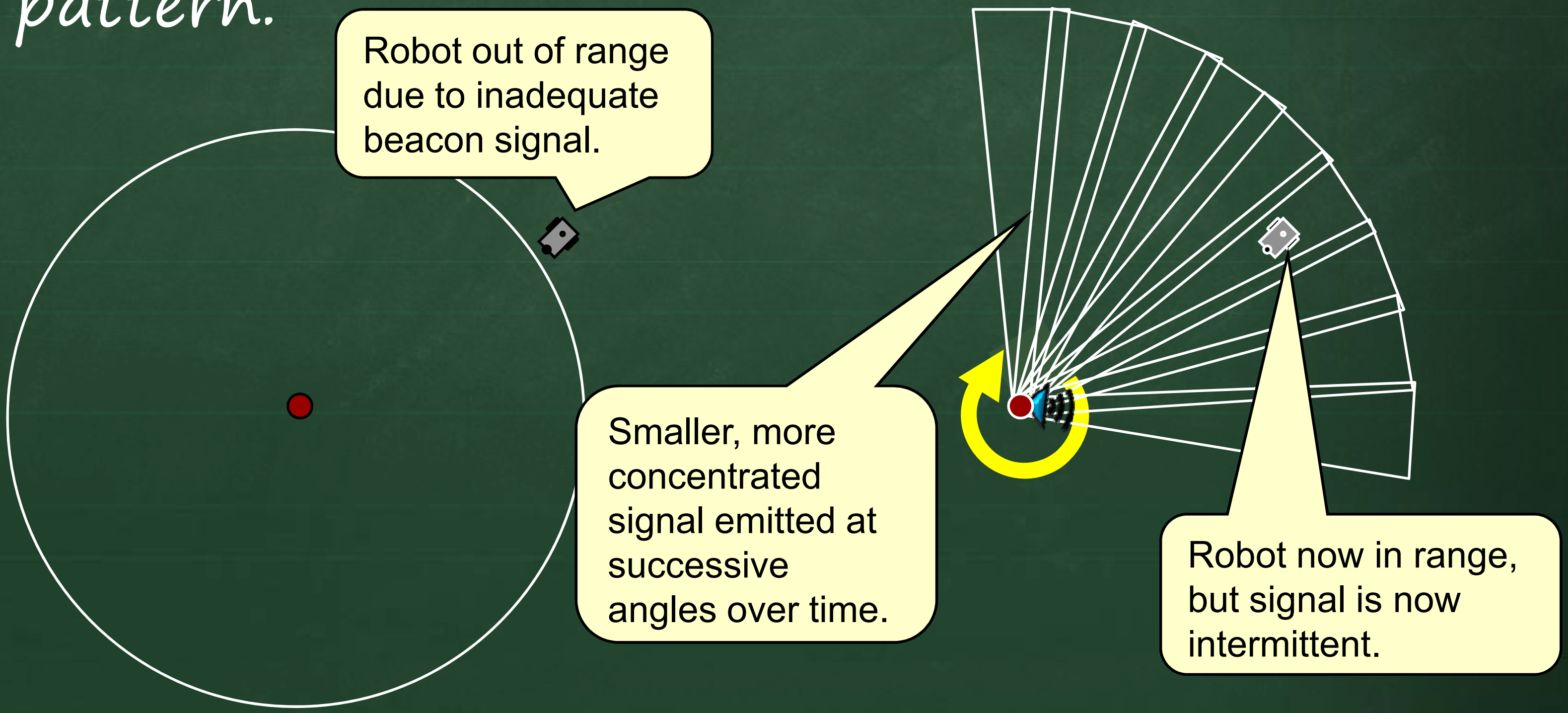
$$c \cdot \sin(\pi - \theta_3 + \theta_1 - \delta) / \sin(\theta_3 - \theta_1) = a \cdot \sin(\phi) / \sin(\theta_1 - \theta_2)$$
$$b \cdot \sin(\sigma) / \sin(2\pi - (\theta_3 - \theta_2)) = a \cdot \sin(\pi - \theta_1 + \theta_2 - \phi) / \sin(\theta_1 - \theta_2)$$
$$c \cdot \sin(\delta) / \sin(\theta_3 - \theta_1) = b \cdot \sin(-\pi + \theta_3 - \theta_2 - \sigma) / \sin(2\pi - (\theta_3 - \theta_2))$$

Triangulation - Angulation

- We can keep reducing this and solve for ϕ, δ & σ
- Working backwards, we can determine β, α & ω
- We can also plug our equations in and solve for e, f & d and then easily compute (x, y).
- There is a lot of math to work out, and it is time consuming, but it can be done.

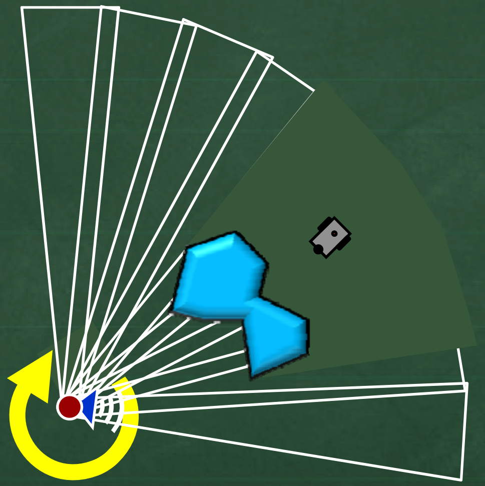
Problems

- Beacons must be extremely powerful to ensure omni-directional transmission over large distances.
- Compromise is to focus the beam and rotate it via some pattern.


Robot out of range due to inadequate beacon signal.
Smaller, more concentrated signal emitted at successive angles over time.
Robot now in range, but signal is now intermittent.

Problems

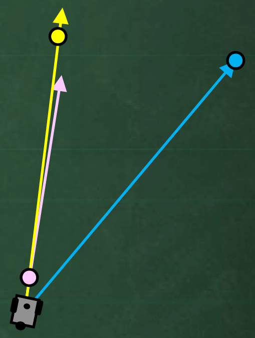
- Beacons may not be visible in some areas due to obstructions from obstacles

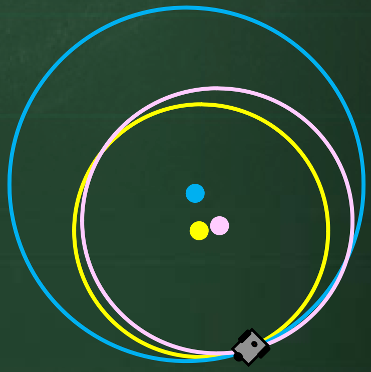


- May need to rely on odometry until reading is available again.

Problems

- Triangulation is sensitive to small angular errors
  - when observed angles are small
  - when measured angles are indistinguishable
  - when robot is far from beacons, can be difficult to determine position accurately





Summary

- You should now understand:
  - Some of the issues behind wheeled robot mobility
  - How to estimate a robot's position using Odometry (dead-reckoning)
  - How to predict a robot's motion based on its wheel velocities
  - How to determine a robot's location based on active beacons and grid tracking.
  - How to control and estimate a PropBot's location