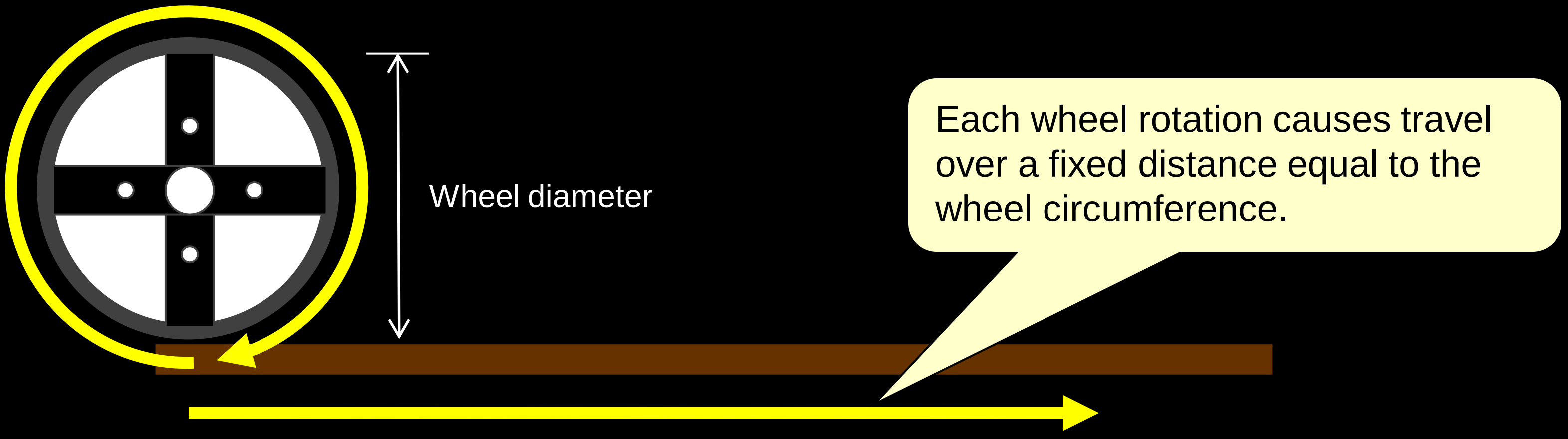
Position Estimation

Velocity

- A robot's velocity (i.e., speed) depends on two things:
  - RPM (Rotations Per Minute) of the wheels
  - Wheel diameter

Velocity

= RPM * Circumference
= RPM * 2π * wheel radius
= RPM * π * wheel diameter



Velocity (continued)

- Imagine that you have a motor spinning at 100RPMs and a wheel diameter of 6cm. How fast is it going ?

  = RPM * π * diameter

  = 100 rpm * π * 6 cm

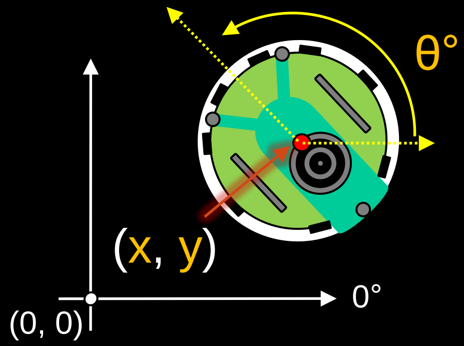
  = 1884.96 cm/minute

  = 31.4 cm/sec

- Larger wheels will increase the velocity, smaller wheels will decrease it.

What is a Pose ?

- A robot's position is its location in the environment:
  - Represented as (x, y) coordinate in 2D or (x, y, z) in 3D
- A robot's orientation is its direction in the environment:
  - Represented as θ° in 2D or (roll°, pitch°, yaw°) in 3D
  - Always with respect to some reference direction such as magnetic North or a horizontal line.
- A robot's pose is a combination of its location and orientation:
  - Represented as (x, y, θ°) in 2D or (x, y, z, roll°, pitch°, yaw°) in 3D

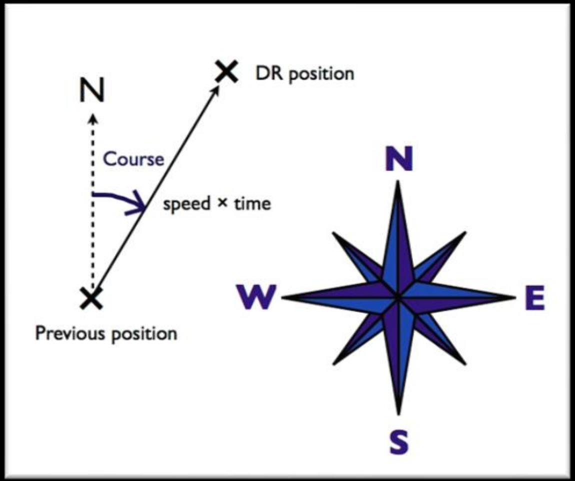


What is Position Estimation ?

- The task of determining the pose of a robot is referred to as pose estimation (also known as position estimation).
- It is an "estimate" because no sensors are accurate enough to give you an exact position or orientation.
- Accuracy of estimate depends on various factors:
  - Quality and reliability of sensors
  - Accuracy of sensor readings
  - Presence of environmental noise and interference
  - Past readings (i.e., accumulative error)
  - Availability of reference data (e.g., maps)

Odometry

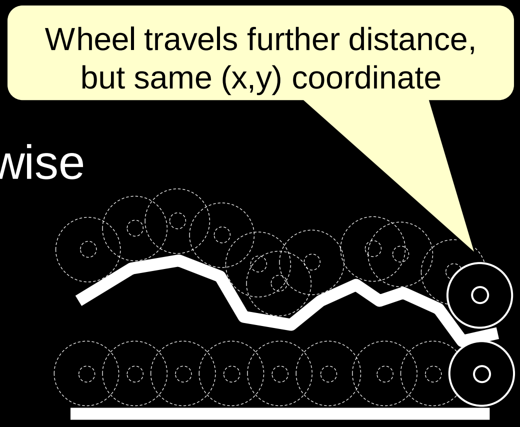
- Odometry is a way of determining a robot's position based on previous known position information given a specific course heading and velocity.
  - Used for years by boats and airplanes
  - Used on many mobile robots



- 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.

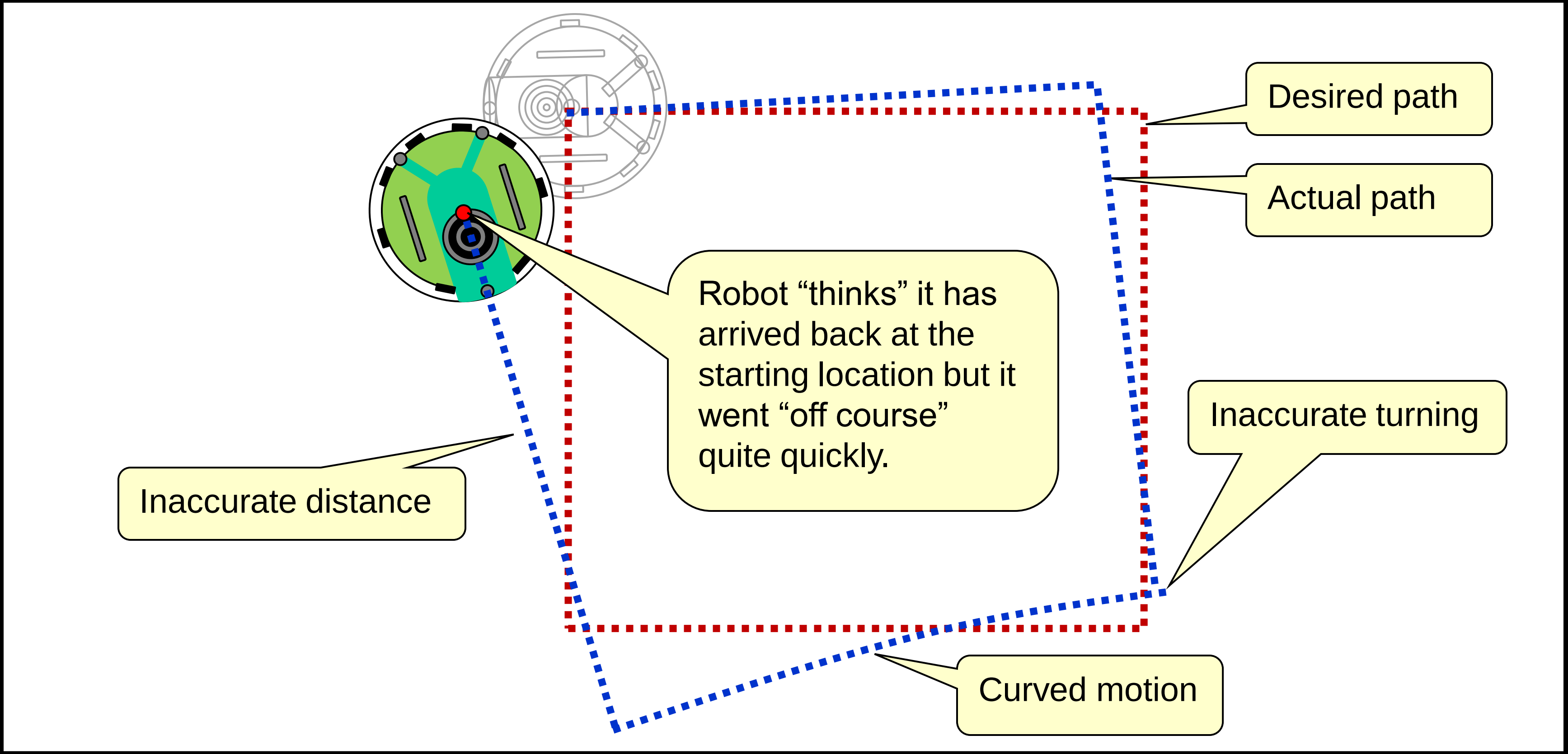
Odometry Errors

- Errors can creep-in due to:
  - Imprecise measurements
    - Actual speed and direction cannot be 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 Errors – The Effect

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



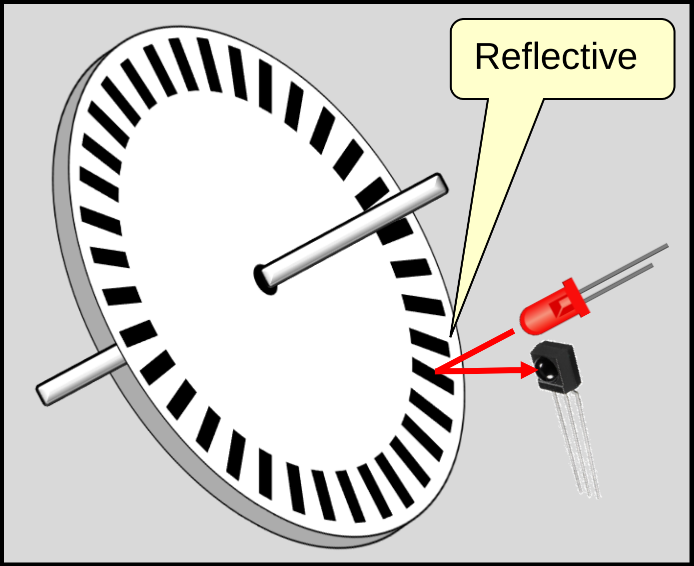
Odometry … There is “Hope”

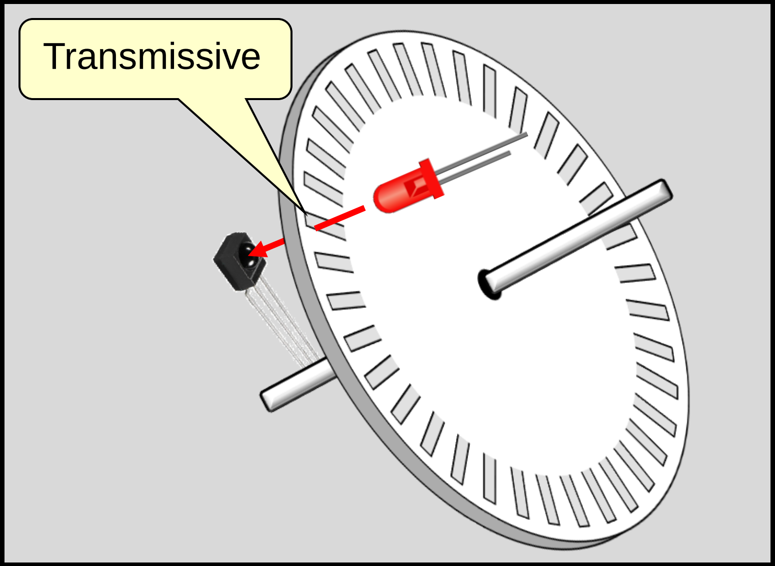
- Theoretically, we can calculate the actual robot’s position as long as:
  1. the robot’s structure is well known, and
  2. the robot’s wheel acceleration/deceleration/velocity or amount of rotation can be accurately measured.
- Various sensors can be used to measure distance, velocity and/or acceleration:
  - Optical encoders (on per wheel)
  - Doppler sensors (usually ultrasonic)
  - Inertial Measurement Units (IMU)

Very popular for wheeled robots … and inexpensive.

Optical Encoders

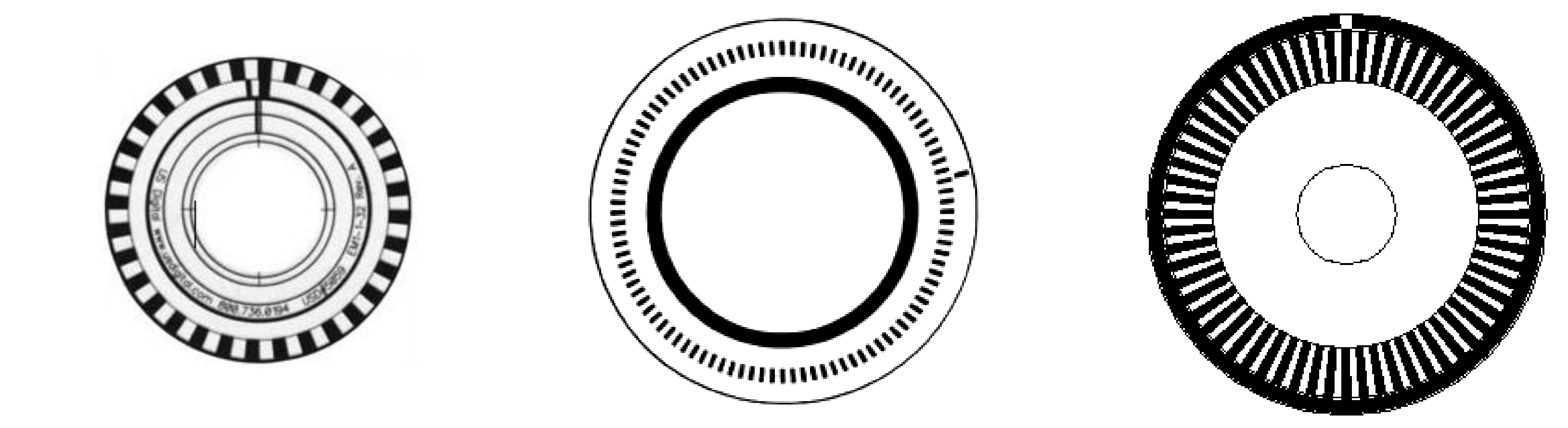
- ***Optical encoders*** are devices used to measure angular position, velocity or amount of rotation.
  - A focused beam of light aimed at a photodetector which is periodically interrupted by an opaque or transparent pattern on a rotating disk attached to the shaft of the wheel.





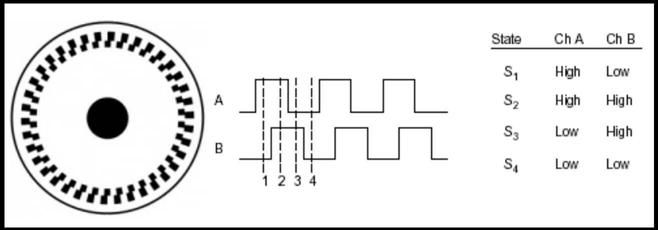
Incremental Optical Encoders

- Disks have evenly spaced slots around border which indicate accuracy:



- Can measure velocity and infer relative position.

- Two subtypes: (1) single-channel (as shown above) or (2) phase-quadrature (allows double position and better accuracy)



Measuring Distance

- If robot is moving straight ahead, simply count encoder ticks to determine how far it travelled.

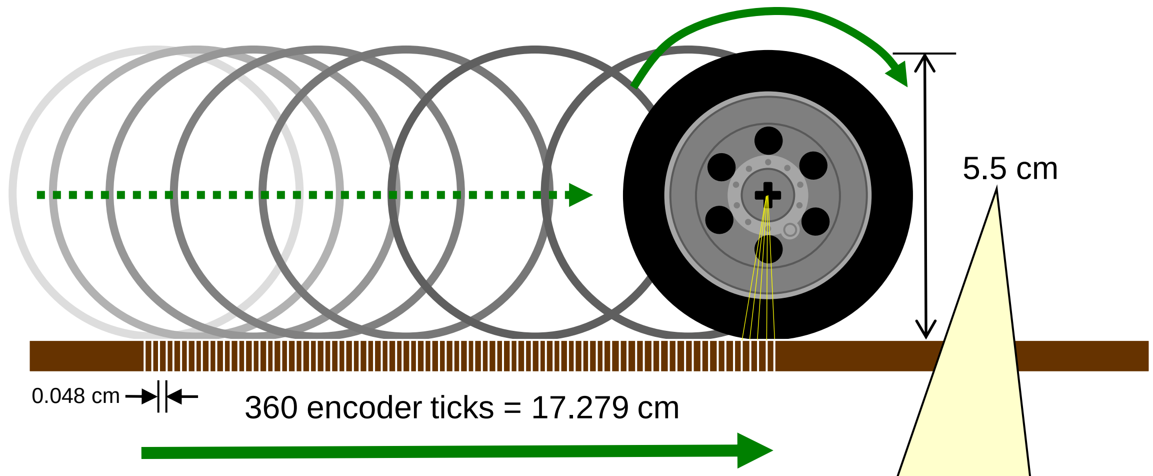
Distance traveled per tick:

$$= \frac{(\text{Wheel Circumference})_{cm}}{360_{ticks}}$$

$$= 5.5_{cm}\ \pi / 360$$

$$= 0.048_{cm}$$

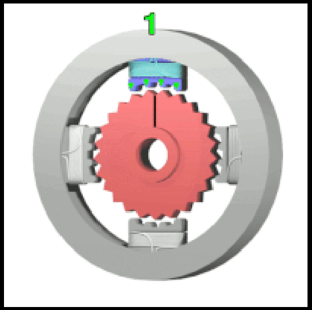
Calculation assumes an encoder with 360 ticks per revolution, but every robot may differ.



Actual wheel diameter can vary (e.g., **± 0.1cm**), depending on the particular wheel, its age, the weight of the robot, the placement of the wheel, etc...

E-Puck - Measuring Distance

- The GcTronic E-puck uses stepper motors instead of encoders.
  - Motor can move in small increments, so position does not need to be measured with an encoder.



- E-puck has 50:1 reduction gear, so provides accuracy of about 0.013cm per motor step.

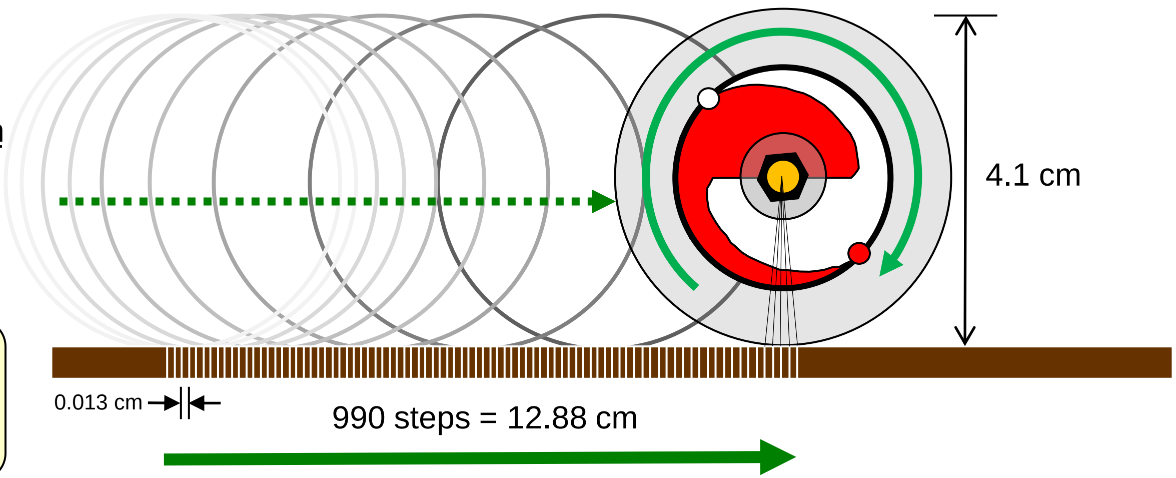
Distance traveled per step:

$$= \frac{(\text{Wheel Circumference})_{cm}}{\sim 990_{steps/rotation}}$$

$$\sim= \pi\ 4.1_{cm} / 990$$

$$\sim= 0.013_{cm}$$

Estimated geared-down steps per full wheel rotation.



E-Puck – Forward Travel Calc.

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

static final double  WHEEL_RADIUS   = 2.05; // cm

// Get the wheel position sensors
PositionSensor leftEncoder = robot.getPositionSensor("left wheel sensor");
PositionSensor rightEncoder = robot.getPositionSensor("right wheel sensor");
leftEncoder.enable(TimeStep);
rightEncoder.enable(TimeStep);

// Read number of radians that the wheel has turned since it started
double previousLeftReading = 0; // start at zero

// MOVE THE ROBOT A BIT
// ...

double leftReading = leftEncoder.getValue();  // value is in radians

// Calculate the distance that the wheel has travelled since the last time it moved
double distance = (leftReading - previousLeftReading) * WHEEL_RADIUS;

previousLeftReading = leftReading; // Get ready for next time
```

Sensor gives number of radians turned by wheel since it started.

distance = (radiansTurned$_{rad}$ / 2π) * wheelCircumference
= ((leftReading – previousLeftReading) / 2π) * (2π· WHEEL_RADIUS)
= (leftReading – previousLeftReading) * WHEEL_RADIUS

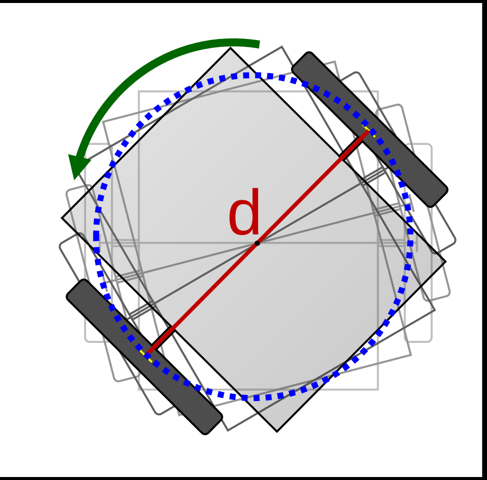
Spin Angle Calculation

- If a 2-wheel differential drive robot spins, we can also count encoder ticks (or motor steps) to determine how many degrees it turned.

  – Each wheel follows the outline of a circle C centered at midpoint between wheels.

  - Circumference of C is defined by distance d between both wheels (i.e., axel length).

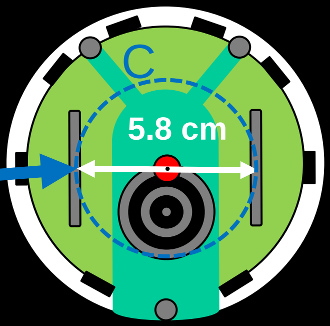
  – The number of degrees turned by the robot as it spins will depend on the portion (or %) of the circumference that is traced out.



E-Puck - Spin Angle Calculation

- Consider the GcTronic E-Puck robot with distance between wheels being 5.8_{cm}.
  - Spinning is centered around a circle C with diameter of 5.8_{cm} and circumference of

    $\pi * 5.8_{cm} = 18.221_{cm}$



  - Use getValue() on a wheel position sensor to get # radians that wheel has turned and therefore know amount of travel (i.e., distance) along C's circumference: $VALUE_{rad} * 2.05_{cm}$

wheel radius

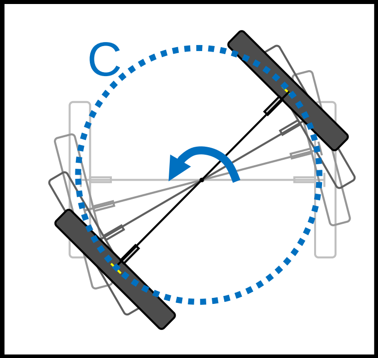
  - Divide C's circumference by this amount as follows:

    % of travel on C's perimeter = $18.221_{cm} / (2.05_{cm} * VALUE_{rad})$

    Angle turned = (% of travel on C's perimeter * 2π) $_{rad}$

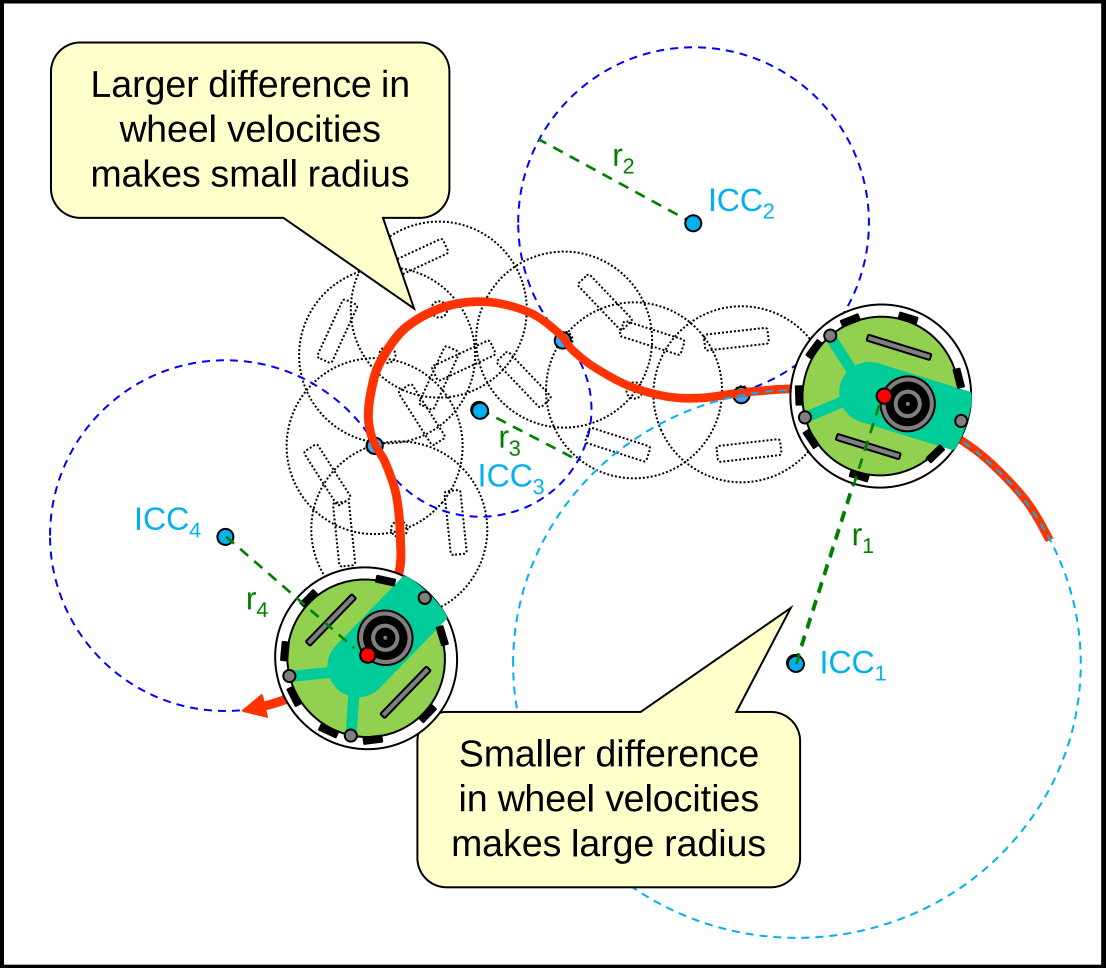
    = (% of travel on C's perimeter * 360)$^{\circ}$

Amount that robot has turned



Kinematics

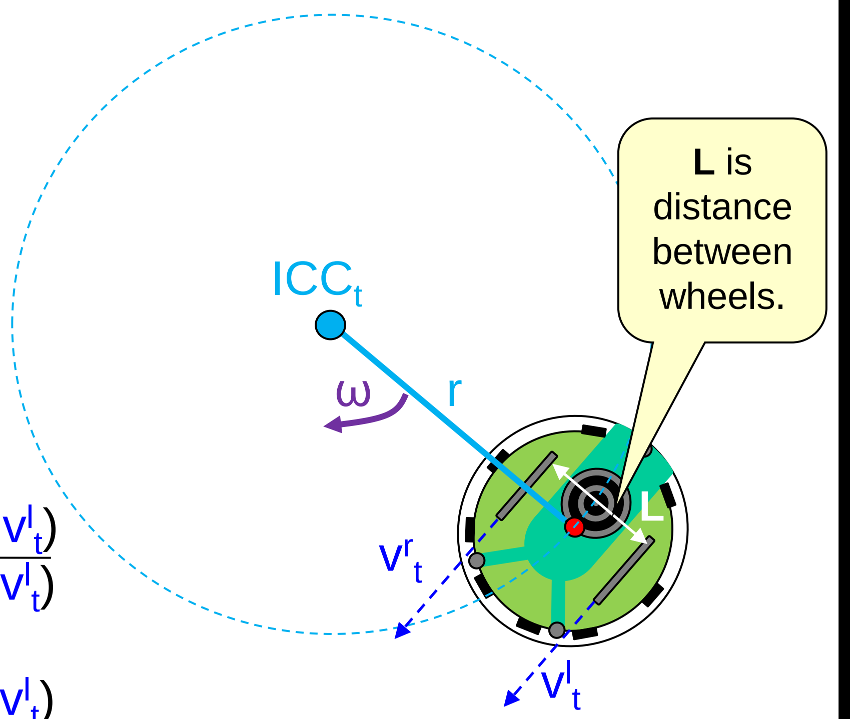
- Recall that the ***instantaneous center of curvature*** (ICC) is the point around which each wheel of the robot makes a circular course.



- ICC changes over time as a function of the individual wheel velocities

Kinematics

- Assume that at each instance of time, t, 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 and ω can both be calculated w.r.t. distance L between wheels and their velocities at time t.



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

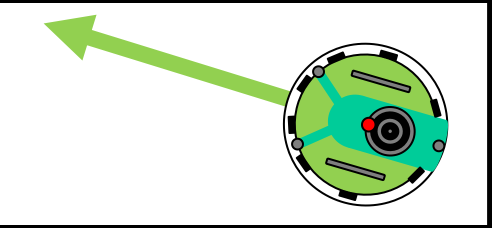
$$\omega = \frac{(v^r_t - v^l_t)}{L}$$

Unit is radians per sec

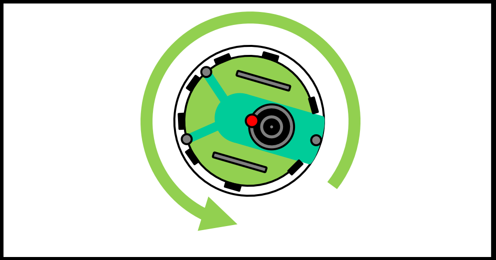
Kinematics

- There are two special cases:

1. When $v^l_t = v^r_t$, then r is infinite and the robot moves in a straight line.



2. When $v^l_t = -v^r_t$, then r is zero and the robot spins (i.e., rotates in place).



- Differential drive robots are very sensitive to the velocity differences between the two wheels:
  - It is hard to move in a perfectly straight line.
  - It is hard to spin exactly in the same location.

Forward Kinematics

- Consider the general ***forward kinematics*** problem:

  Given:

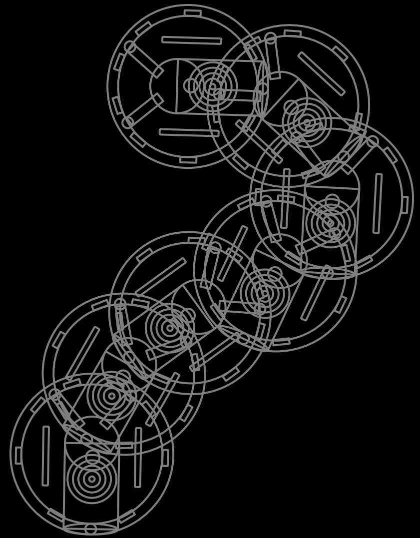
  robot location (x_t, y_t) and orientation θ_t at time t

  Find:

  robot location ($x_{t+\delta}$, $y_{t+\delta}$) and orientation $\theta_{t+\delta}$ at time $t+\delta$

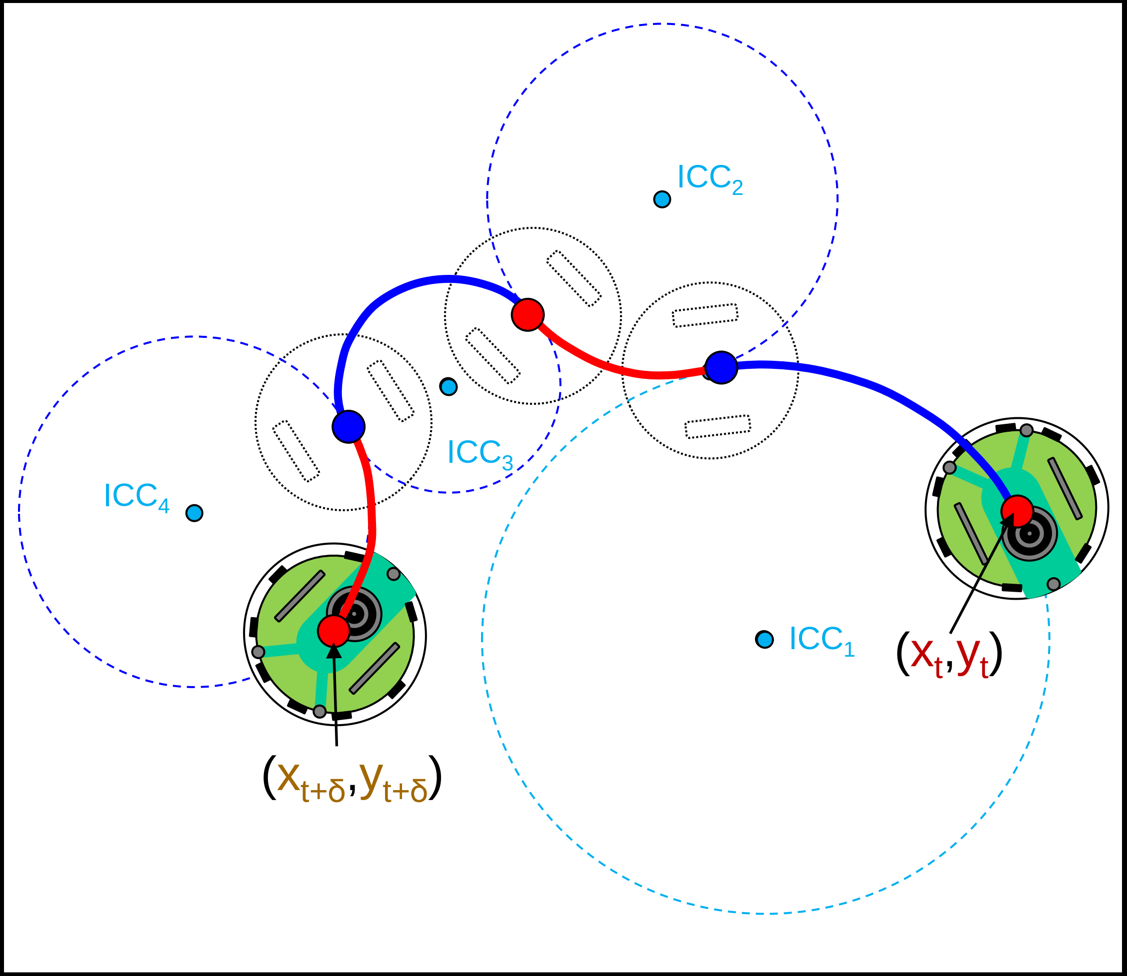
δ is an arbitrary number of seconds in the future.

- This is not a straight-forward task
  - Robot may move, turn, spin arbitrarily
  - Robot may speed-up, slow-down and stop



Forward Kinematics

- Problem is manageable if we break down path of robot into pieces that have constant motor speed and deal with each separately.
- Need to compute points each time the motor speeds change.



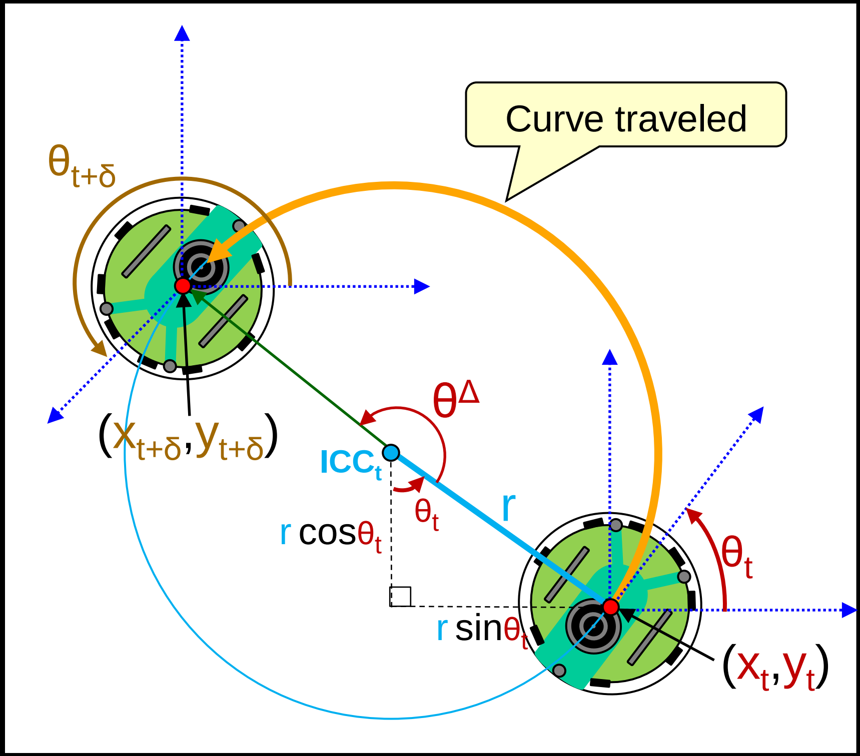
Forward Kinematics

- Consider one such portion of the path.

- We must determine …

  $\mathrm{ICC}_t = (X_{icc}, Y_{icc})$

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



- But how do we determine radius r since we do not know the wheel velocities ?

- We can compute r in terms of encoder ticks ...

Forward Kinematics

- Wheels trace out circles with different circumferences:
  - Dist. traveled by left wheel:

    $D_L = |r_L * \theta^{\Delta}|$

  - Dist. traveled by right wheel:

    $D_R = |(r_L + L) * \theta^{\Delta}|$

    $= |D_L + L * \theta^{\Delta}|$

  - We can re-arrange to get these:

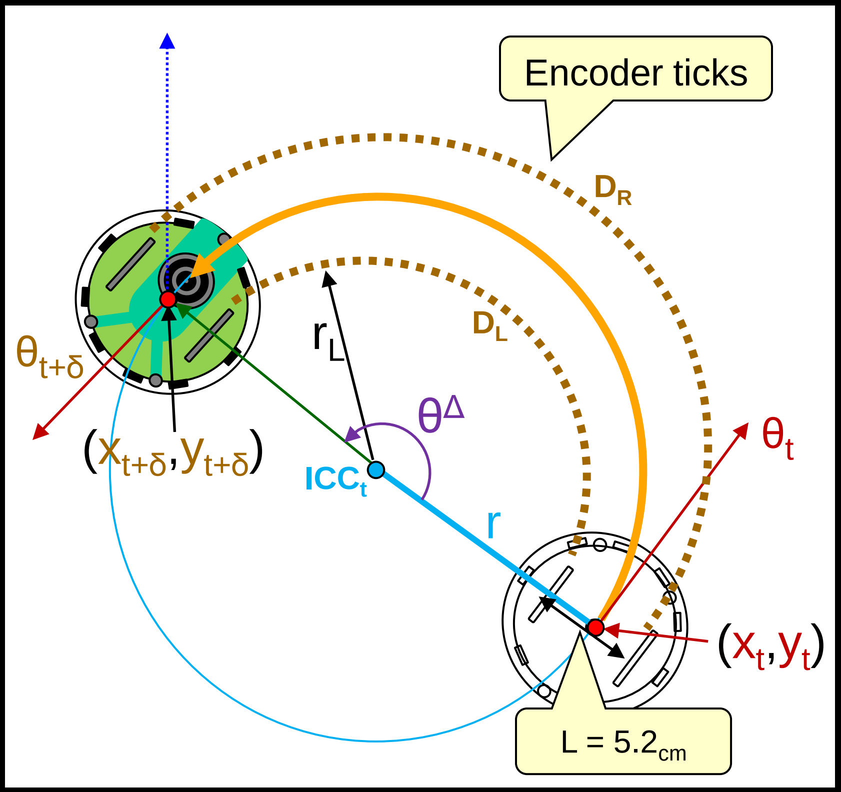
    May be negative

    $\theta^{\Delta} = (D_R - D_L) / L$

    $r_L = LD_L / (D_R - D_L)$

  - And so,

    $r = r_L + L/2 = LD_L / (D_R - D_L) + L/2$

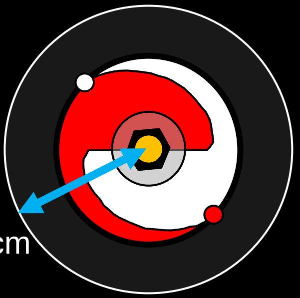


Forward Kinematics

- Recall that getValue() gives radians travelled for an E-Puck wheel. Therefore, D_R and D_L can be determined by calling that function for each wheel:

  D_R = $rightReading_{rad}$ * radius = 2.05_{cm} * rightReading

  D_L = $leftReading_{rad}$ * radius = 2.05_{cm} * leftReading



- We can compute r now as follows:

  $r = L * (D_L / (D_R – D_L) + ½)$

  $= 5.8_{cm} * (2.05_{cm} * leftReading / (2.05_{cm} * rightReading - 2.05_{cm} * leftReading) + ½)$

  $= [5.8 * (leftReading / (rightReading - leftReading)) + 2.9]_{cm}$

Forward Kinematics

- Recall that the amount of turning that happened is:

$$\begin{aligned}
\theta^{\Delta} &= (D_R - D_L) / L \\
&= (2.05_{cm} * \text{rightReading} - 2.05_{cm} * \text{leftReading}) / L \\
&= (2.05_{cm} * (\text{rightReading} - \text{leftReading})) / 5.8_{cm} \\
&= (\text{rightReading} - \text{leftReading}) * 0.35344828_{radians} \\
&= (\text{rightReading} - \text{leftReading}) * 20.2510945^{\circ}
\end{aligned}$$

- Now that we have the ICC radius r and the change in angle θ^{Δ}, we can determine the new location and angle by applying some "nifty" formulas.

- Assume that we know (x_t, y_t) and θ_t at time t and that we want to compute $(x_{t+\delta}, y_{t+\delta})$ and $\theta_{t+\delta}$ at time $t+\delta$ for some δ number of seconds.

Forward Kinematics – Curving

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

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

- Since $X_{ICC} = x_t - r \cdot \sin\theta_t$ and $Y_{ICC} = y_t + r \cdot \cos\theta_t$ … then:

$$\left.\begin{aligned} x_{t+\delta} &= r \cdot \cos\theta^{\Delta} \cdot \sin\theta_t + r \cdot \cos\theta_t \cdot \sin\theta^{\Delta} + x_t - r \cdot \sin\theta_t \\ y_{t+\delta} &= r \cdot \sin\theta^{\Delta} \cdot \sin\theta_t - r \cdot \cos\theta_t \cdot \cos\theta^{\Delta} + y_t + r \cdot \cos\theta_t \\ \theta_{t+\delta} &= \theta_t + \theta^{\Delta} \end{aligned}\right\}$$

where,

$$r = [5.8 * (\text{leftReading} / (\text{rightReading} - \text{leftReading})) + 2.9]_{cm}$$

$$\theta^{\Delta} = (\text{rightReading} - \text{leftReading}) * 20.2510945\,^{\circ}$$

So … we just compute these equations each time the robot's wheels speed changes.

Forward Kinematics – Straight

- Straight forward movement is a special case
  - When (rightReading == leftReading) then $r = \infty$
    - Therefore, ICC equation cannot be used
- The math is even simpler:

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

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

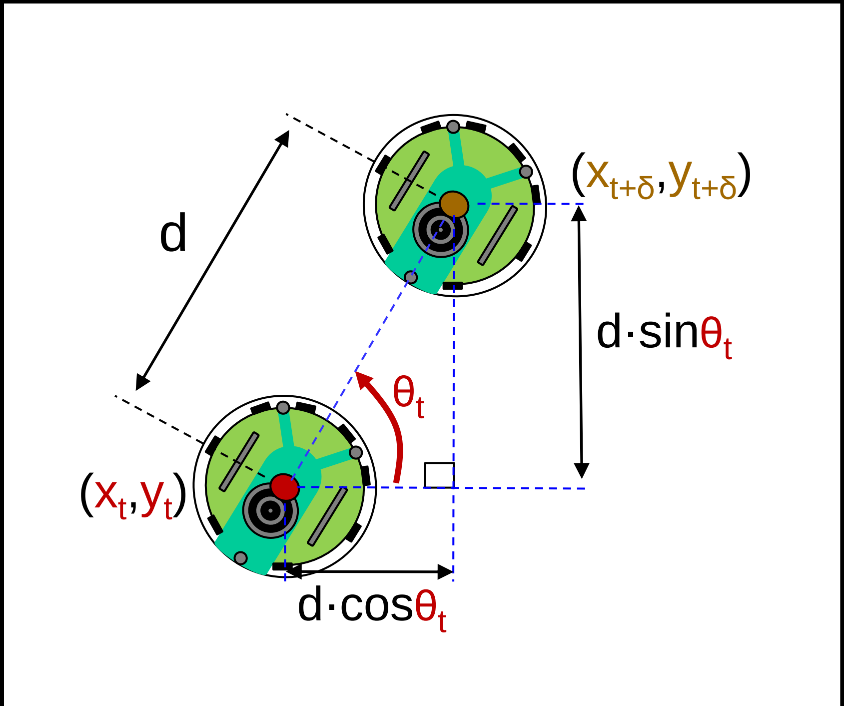
$$\theta_{t+\delta} = \theta_t$$

Angle does not change.

where,

$$d = \text{leftReading} * 2.05_{cm}$$

Can be either left or right sensor value.



Forward Kinematics – Spinning

- Spinning is also a special case
  - when (rightReading = - leftReading) then r = 0 … ICC equation cannot be used
- The math is also simple:

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

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

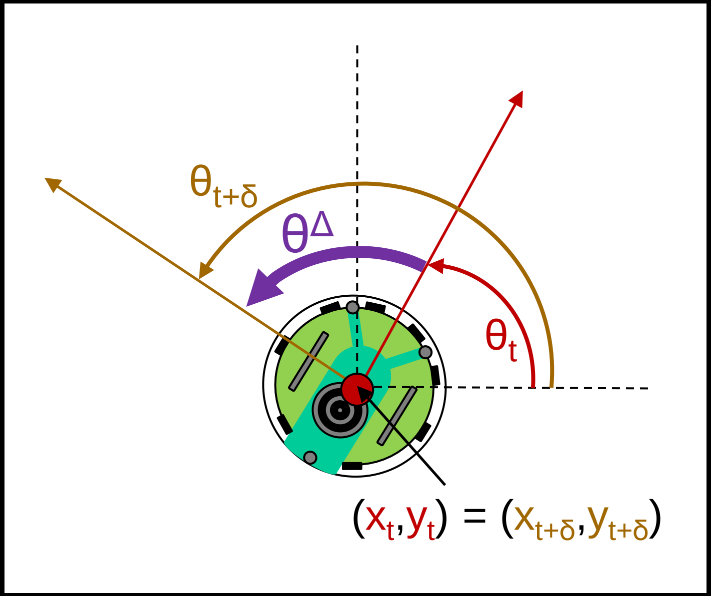
Location does not change.

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

where

$$\theta^{\Delta} = (\text{rightReading} - \text{leftReading}) * 20.2510945^{\circ}$$

Difference between right and left sensor values



Start the Lab ...