CS W4733 NOTES - Differential Drive Robots

Note: these notes were compiled from Dudek and Jenkin, *Computational Principles of Mobile Robotics*.

# 1 Differential Drive Kinematics

Many mobile robots use a drive mechanism known as differential drive. It consists of 2 drive wheels mounted on a common axis, and each wheel can independently being driven either forward or backward.

While we can vary the velocity of each wheel, for the robot to perform rolling motion, the robot must rotate about a point that lies along their common left and right wheel axis. The point that the robot rotates about is known as the ICC - *Instantaneous Center of Curvature* (see figure 1).

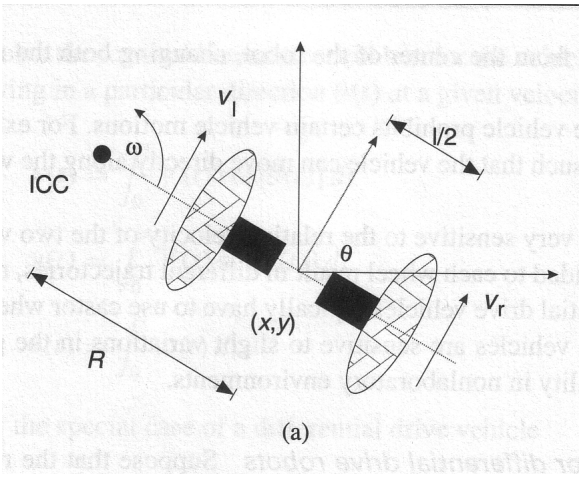



Figure 1: Differential Drive kinematics (from Dudek and Jenkin, *Computational Principles of Mobile Robotics*.

By varying the velocities of the two wheels, we can vary the trajectories that the robot takes. Because the rate of rotation $\omega$ about the ICC must be the same for both wheels, we can write the following equations:

$$\omega\,(R + l/2) = V_r \quad (1)$$
$$\omega\,(R - l/2) = V_l \quad (2)$$

where $l$ is the distance between the centers of the two wheels, $V_r$ , $V_l$ are the right and left wheel velocities along the ground , and R is the signed distance from the ICC to the midpoint between the wheels. At any instance in time we can solve for R and $\omega$:

$$R = \frac{l}{2}\frac{V_l + V_r}{V_r - V_l}\,;\ \omega = \frac{V_r - V_l}{l}; \tag{3}$$

There are three interesting cases with these kinds of drives.

1. If $V_l = V_r$, then we have forward linear motion in a straight line. $R$ becomes infinite, and there is effectively no rotation - $\omega$ is zero.
2. If $V_l = -V_r$, then $R = 0$, and we have rotation about the midpoint of the wheel axis - we rotate in place.
3. If $V_l = 0$, then we have rotation about the left wheel. In this case $R = \frac{l}{2}$. Same is true if $V_r = 0$.

Note that a differential drive robot cannot move in the direction along the axis - this is a singularity. Differential drive vehicles are very sensitive to slight changes in velocity in each of the wheels. Small errors in the relative velocities between the wheels can affect the robot trajectory. They are also very sensitive to small variations in the ground plane, and may need extra wheels (castor wheels) for support.

# 2 Forward Kinematics for Differential Drive Robots

In figure 1, assume the robot is at some positon $(x, y)$, headed in a direction making an angle $\theta$ with the $X$ axis. We assume the robot is centered at a point midway along the wheel axle. By manipulating the control parameters $V_l$ , $V_r$, we can get the robot to move to different positions and orientations. (note: $V_l$ , $V_r$) are wheel velocities along the ground).

Knowing velocities $V_l, V_r$ and using equation 3, we can find the ICC location:

$$ICC = [x - R\,sin(\theta)\,,\ y + R\,cos(\theta)] \tag{4}$$

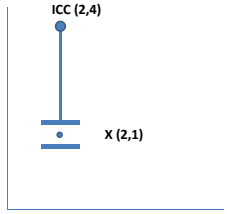
and at time $t + \delta t$ the robot's pose will be:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} cos(\omega\delta t) & -sin(\omega\delta t) & 0 \\ sin(\omega\delta t) & cos(\omega\delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ \omega\delta t \end{bmatrix} \tag{5}$$

This equation simply describes the motion of a robot rotating a distance $R$ about its ICC with an angular velocity of $\omega$.

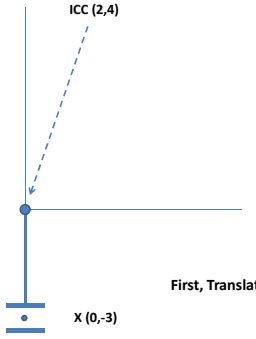
Refer to figure 2. Another way to understand this is that the motion of the robot is equivalent to 1) translating the ICC to the origin of the coordinate system, 2) rotating about the origin by an angular amount $\omega\,\delta t$, and 3) translating back to the ICC.

## Forward Kinematics for Differential Drive Robot



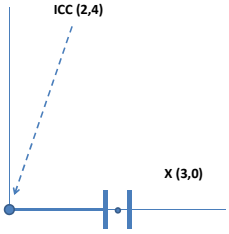

Example: Differential Drive Robot. Rotate about ICC 90 degrees.
How do we know where the robot ends up?

## Forward Kinematics for Differential Drive Robot



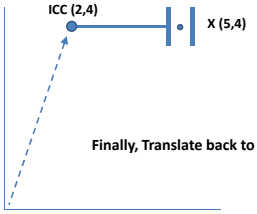

First, Translate ICC to origin

## Forward Kinematics for Differential Drive Robot




Then, Rotate by 90 degrees about Z axis

## Forward Kinematics for Differential Drive Robot




Finally, Translate back to original ICC

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} cos(\omega\delta t) & -sin(\omega\delta t) & 0 \\ sin(\omega\delta t) & cos(\omega\delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ \omega\delta t \end{bmatrix}$$

| Transformed Point X,Y | Rotation by wdt about Z axis | Translate ICC to origin | Translate ICC Back to original location |
|---|---|---|---|

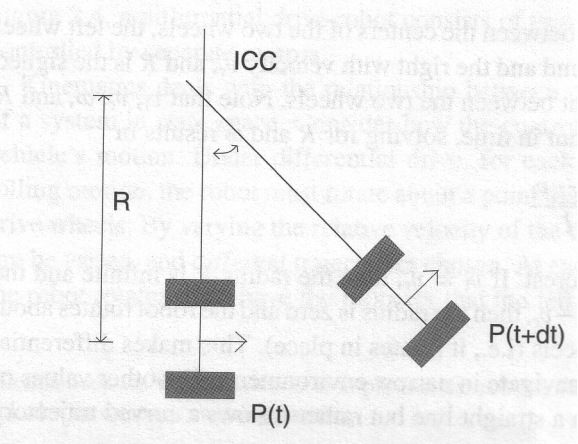



Figure 2: Forward kinematics for differential robot

# 3 Inverse Kinematics of a Mobile Robot

In general, we can describe the positon of a robot capable of moving in a particular direction $\Theta(t)$ at a given velocity $V(t)$ as:

$$\begin{aligned} x(t) &= \int_0^t V(t)cos[\theta(t)]dt \\ y(t) &= \int_0^t V(t)sin[\theta(t)]dt \\ \Theta(t) &= \int_0^t \omega(t)dt \end{aligned}$$

For the special case of a differential drive robot such as the GoPiGo, the equations become:

$$\begin{aligned} x(t) &= \frac{1}{2}\int_0^t [v_r(t) + v_l(t)]cos[\theta(t)]dt \\ y(t) &= \frac{1}{2}\int_0^t [v_r(t) + v_l(t)]sin[\theta(t)]dt \\ \Theta(t) &= \frac{1}{l}\int_0^t [v_r(t) - v_l(t)])dt \end{aligned}$$

A related question is: How can we control the robot to $reach$ a given configuration $(x, y, \theta)$ - this is known as the inverse kinematics problem.

Unfortunately, a differential drive robot imposes what are called *non-holonomic* constraints on establishing its position. For example, the robot cannot move laterally along its axle. A similar non-holonomic constraint is a car that can only turn its front wheels. It cannot move directly sidewise, as parallel parking a car requires a more complicated set of steering maneuvers. So we cannot simply specify an arbitrary robot pose $(x, y, \theta)$ and find the velocities that will get us there.

For the special cases of $v_l = v_r = v$ (robot movng in a straight line) the motion equations become:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x + v\,cos(\theta)\delta t \\ y + v\,sin(\theta)\delta t \\ \theta \end{bmatrix} \tag{6}$$

If $v_r = -v_l = v$, then the robot rotates in place and the equations become:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta + 2v\delta t/l \end{bmatrix} \tag{7}$$

This motivates a strategy of moving the robot in a straight line, then rotating for a turn in place, and then moving straight again as a navigation strategy for differential drive robots.

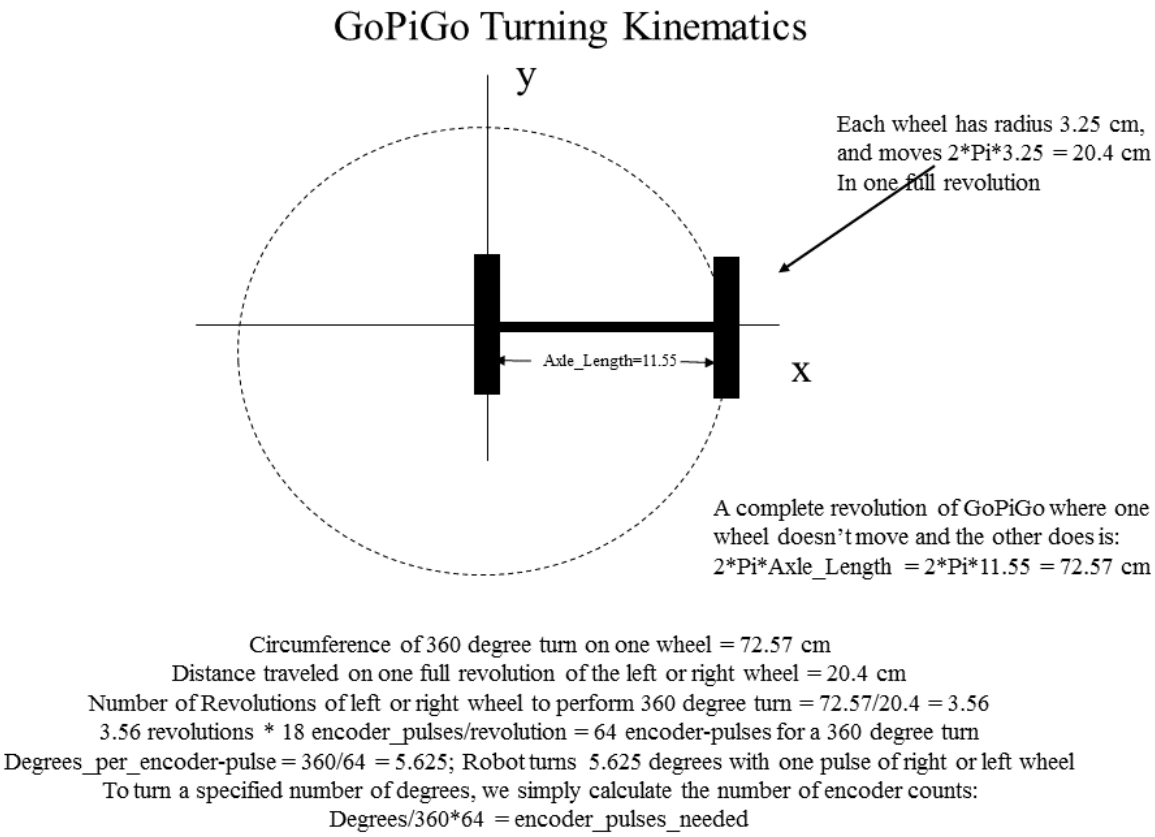



Figure 3: GoPiGo wheel kinematics

# 4 Understanding GoPiGo Kinematics

The GoPiGo's kinematics use two parameters: the wheel radius $R_{wheel}$ ($\sim 3.25$ cm) and the distance between the two drive wheels, the $AxleLength$ ($\sim 11.55$ cm). Each robot has some difference with these numbers, so you may want to check on them.

The number of encoder counts on each wheel is 18: Measuring 18 pulses of the encoder is one full revolution of the wheel.

To move forward a certain distance $D_{forward}$, we need to find out how many encoder counts map to the distance needed. Given the two parameters above, we note that each revolution of a wheel moves the wheel linearly $2\pi R_{wheel}$. If both wheels are moving at the same rate we get forward motion, and the number of wheel revolutions is is $\frac{D_{forward}}{2\pi R_{wheel}}$, and the number of encoder counts to move this distance is just the number of revolutions of the wheels time 18 pulses per revolution:

$$encoder\_counts = \frac{D_{forward}}{2\pi R_{wheel}} \cdot 18$$

Turning: we can turn left by turning only the right wheel, and turn right by turning only the left wheel. We need to compute how many encoder pulses to use to turn either left or right a certain number of degrees. First, we need to calcuate the ratio **DPR: Degrees Per Pulse** for the wheel encoders. To do this, we note that the GoPiGo wheel radius is 3.25 cm, and the linear distance the wheel travels in one revolution is $2 \cdot 3.25\,\pi \;=\; 20.42$ cm. If we only turn one wheel, then the robot makes a circular motion with the circle's circumference being $2 \cdot AxleLength \cdot \pi \;=\; 72.57$ cm. The number of wheel revolutions for a left or right complete circle turn is $72.57/20.4 \;=\; 3.56$, and $3.56 \cdot 18 = 64$ encoder pulses for a full 360 degree turn. Determining the number of encoder pulses for turn less than 360 degrees can be done using just $\frac{Degrees-to-turn}{360} \cdot 64$ encoder pulses on the left wheel (turn clockwise) or right wheel (turn counter-clockwise). The code below is from the Find_Hole programs.

```
from gopigo import *

en_debug=1
## 360 roation is ~64 encoder pulses or 5.625 deg/pulse
## DPR is the Deg:Pulse Ratio or the # of degrees per
##  encoder pulse.
DPR = 360.0/64
WHEEL_RAD = 3.25 # Wheels are ~6.5 cm diameter.

def left_deg(deg=None):
    '''
    Turn chassis left by a specified number of degrees.
    DPR is the #deg/pulse (Deg:Pulse ratio)
    This function sets the encoder to the correct number
     of pulses and then invokes left().
    '''
    if deg is not None:
        pulse= int(deg/DPR)
        enc_tgt(1,0,pulse)
    left()

def right_deg(deg=None):
    '''
    Turn chassis right by a specified number of degrees.
    DPR is the #deg/pulse (Deg:Pulse ratio)
    This function sets the encoder to the correct number
     of pulses and then invokes right().
    '''
    if deg is not None:
        pulse= int(deg/DPR)
        enc_tgt(0,1,pulse)
    right()
```

```
def fwd_cm(dist=None):
    '''
    Move chassis fwd by a specified number of cm.
    This function sets the encoder to the correct number
     of pulses and then invokes fwd().
    '''
    if dist is not None:
        pulse = int(cm2pulse(dist))
        enc_tgt(1,1,pulse)
    fwd()

def bwd_cm(dist=None):
    '''
    Move chassis bwd by a specified number of cm.
    This function sets the encoder to the correct number
     of pulses and then invokes bwd().
    '''
    if dist is not None:
        pulse = int(cm2pulse(dist))
        enc_tgt(1,1,pulse)
    bwd()

def cm2pulse(dist):
    '''
    Calculate the number of pulses to move the chassis dist cm.
    pulses = dist * [pulses/revolution]/[dist/revolution]
    '''
    wheel_circ = 2*math.pi*WHEEL_RAD # [cm/rev] cm traveled per revolution o
    revs = dist/wheel_circ
    PPR = 18 # [p/rev] encoder Pulses Per wheel Revolution
    pulses = PPR*revs # [p] encoder pulses required to move dist cm.
    if en_debug:
        print 'WHEEL_RAD',WHEEL_RAD
        print 'revs',revs
        print 'pulses',pulses
    return pulses
```