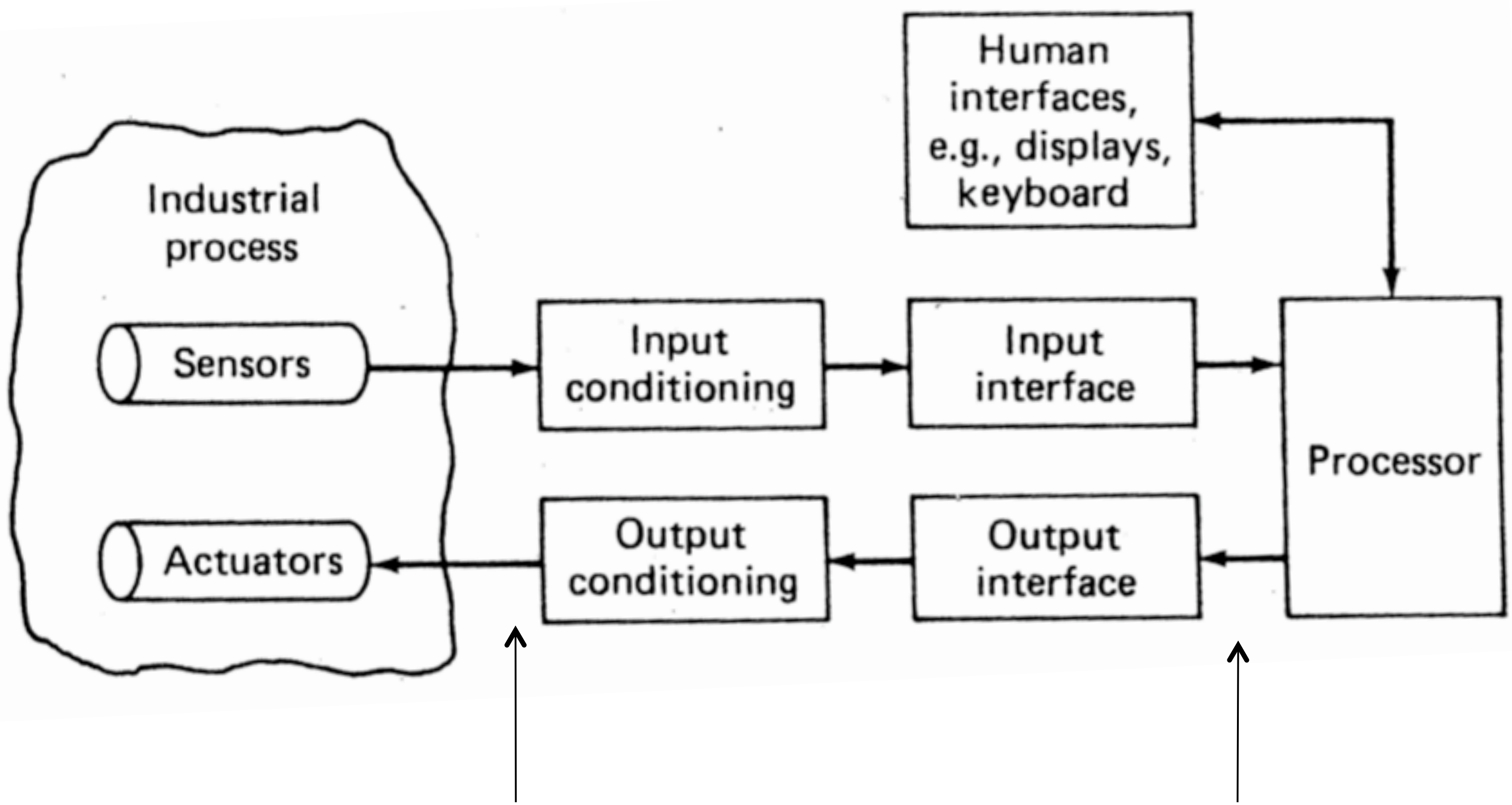
Actuators and acting



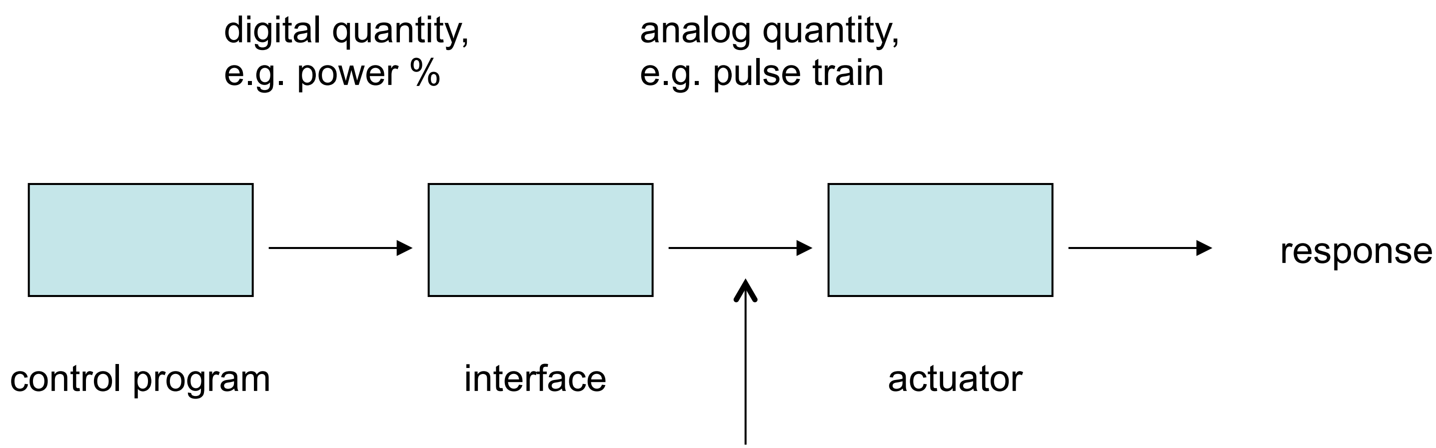
electrical signal

digital signal


NXT

goForward(50)

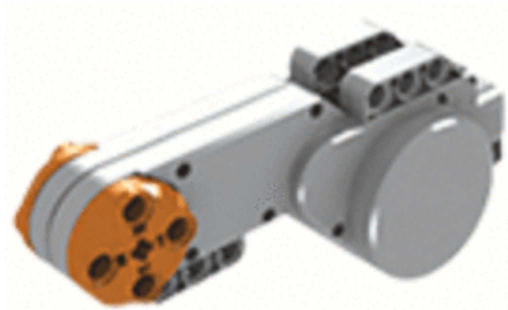
car drives forward



Program controlled power supply

Open loop control

NXT Actuator

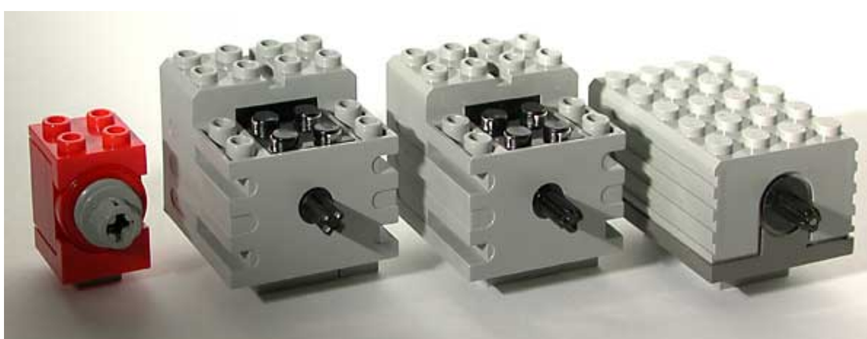


RJ12 connector

6 wire cable

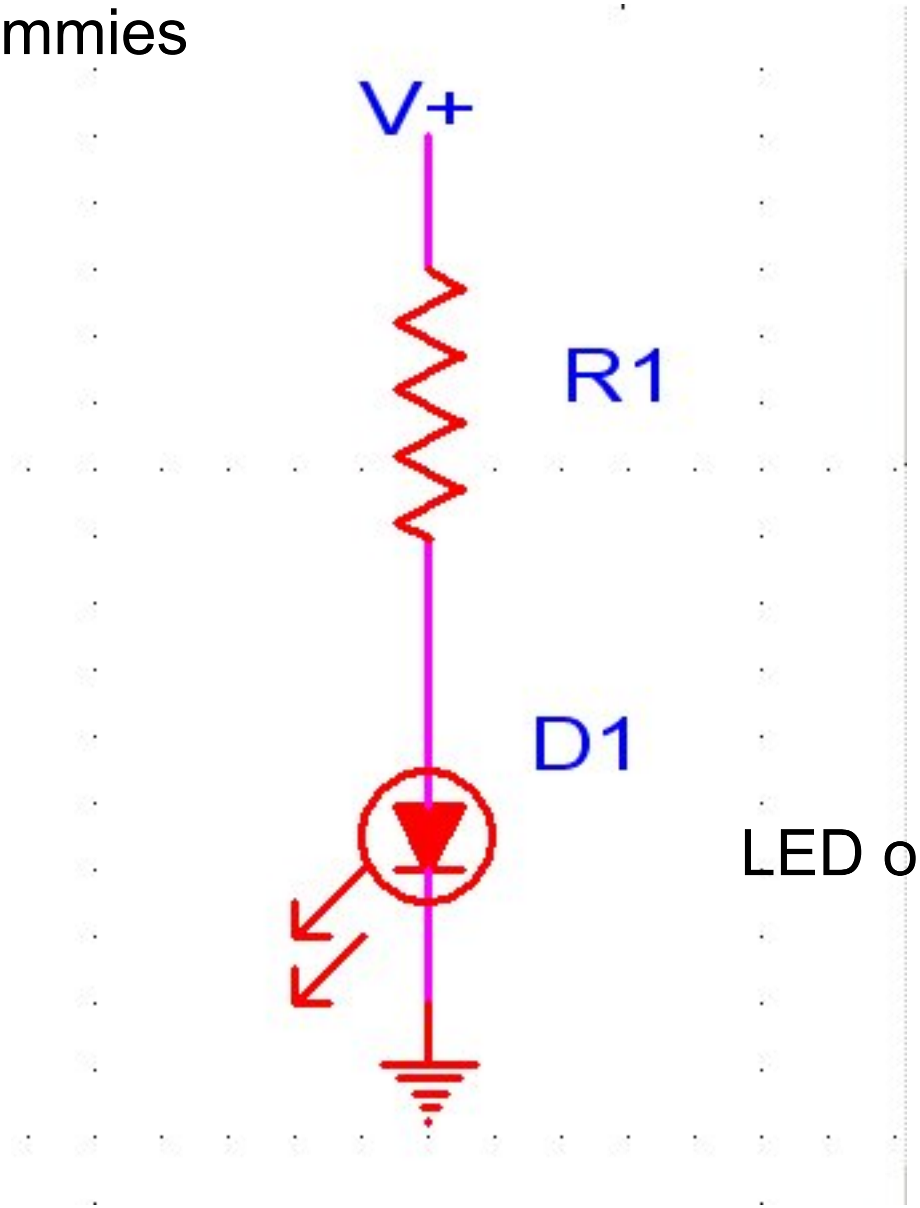
RCX Actuators

2x2 plate connector

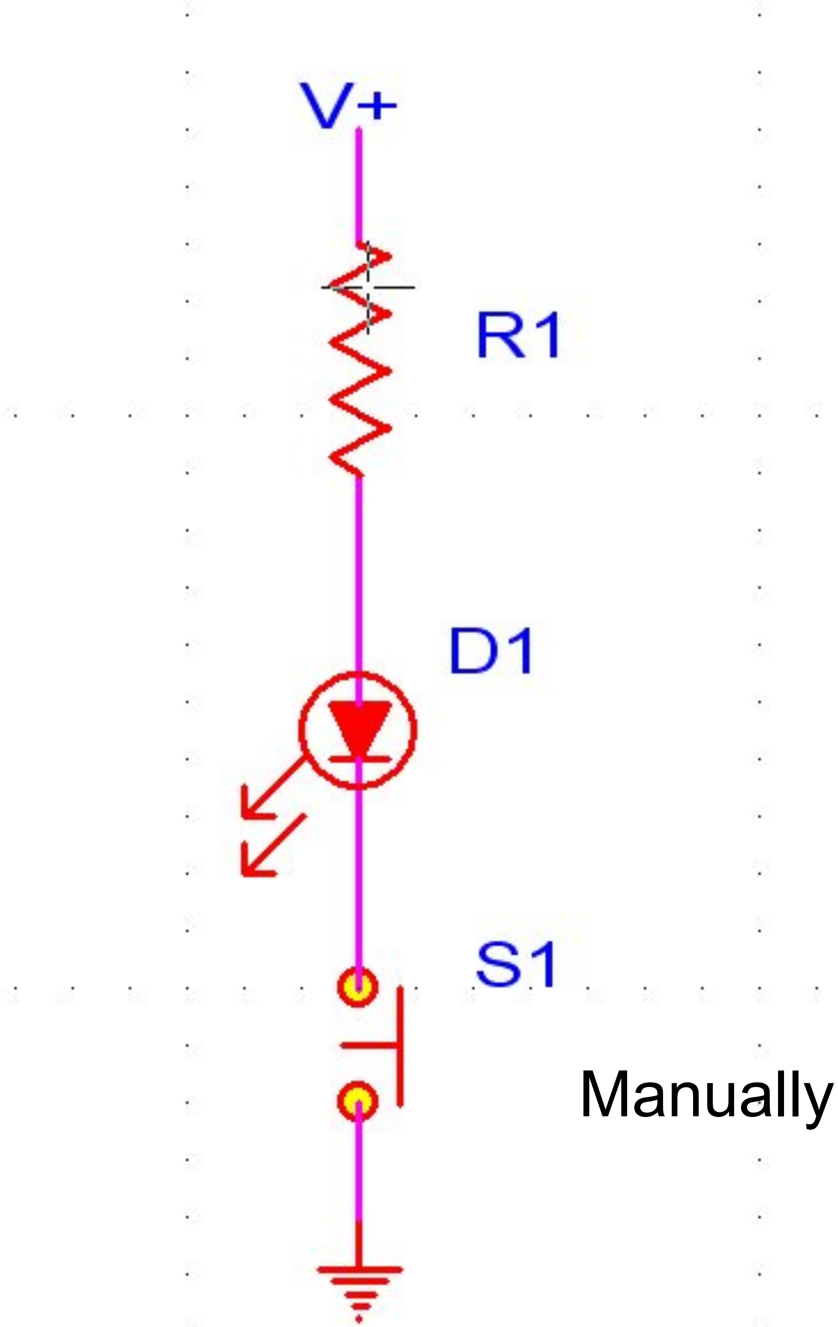


2 wire cable

LED For Dummies

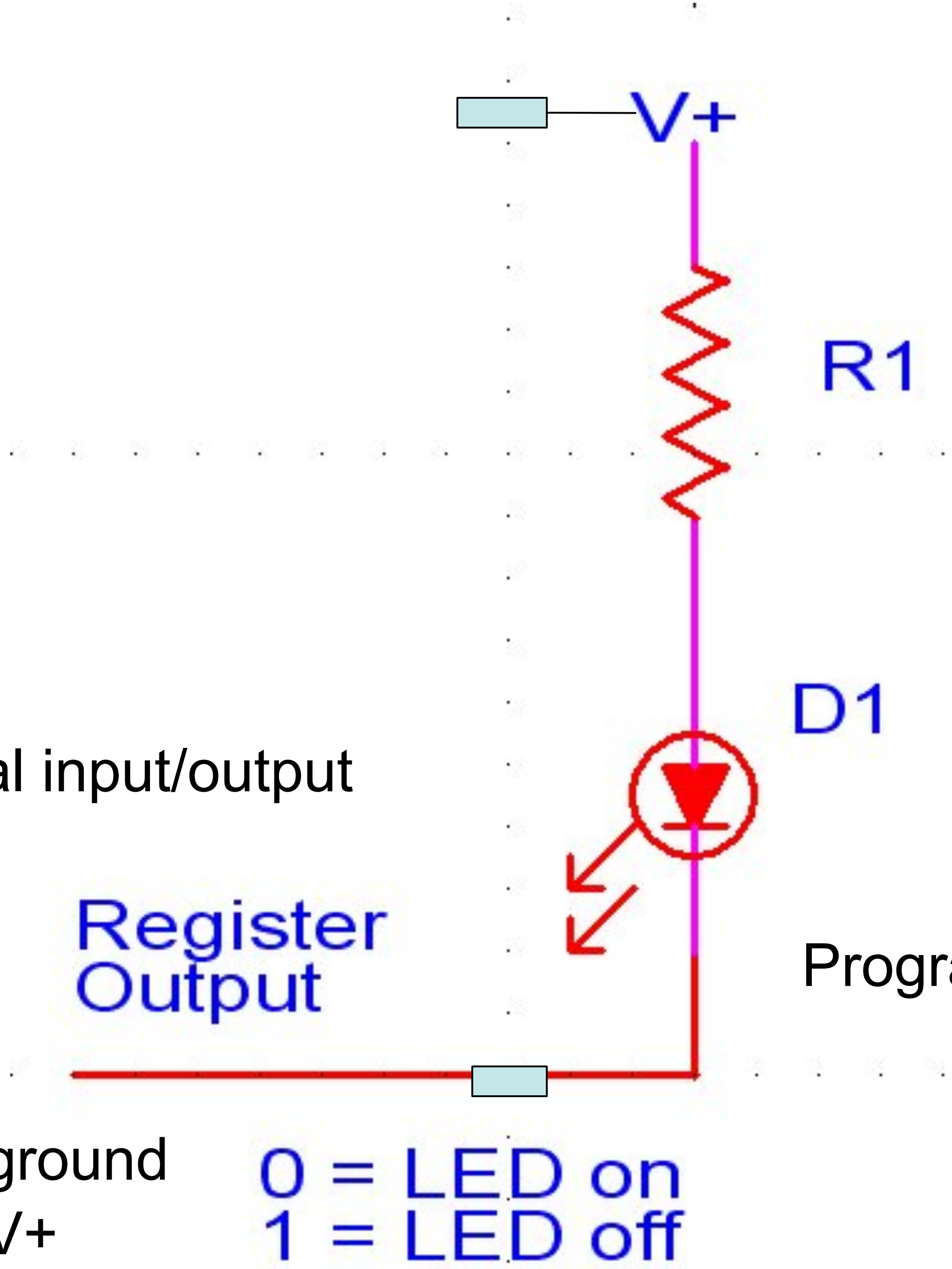


LED on all the time



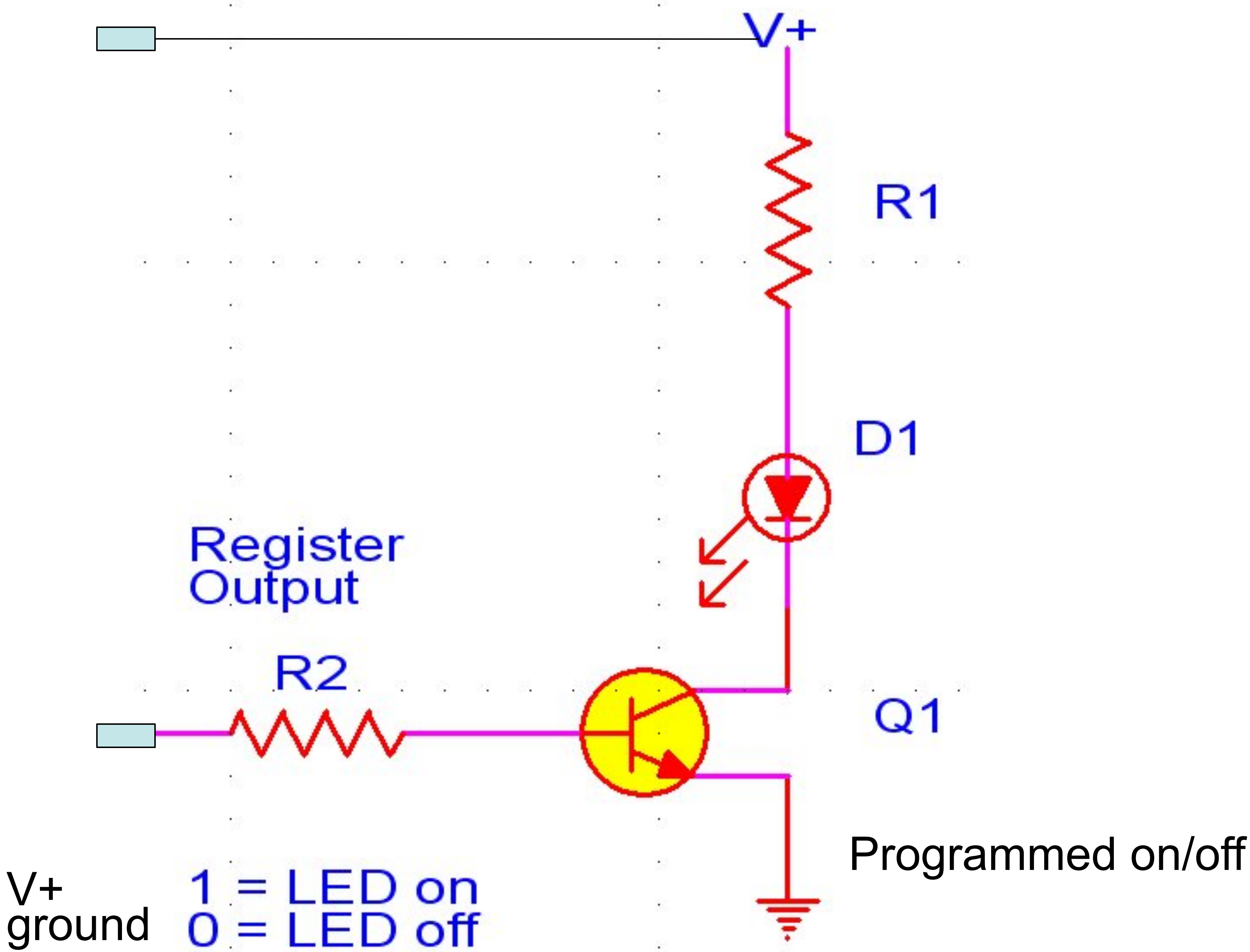
Manually operated switch

GIO general input/output

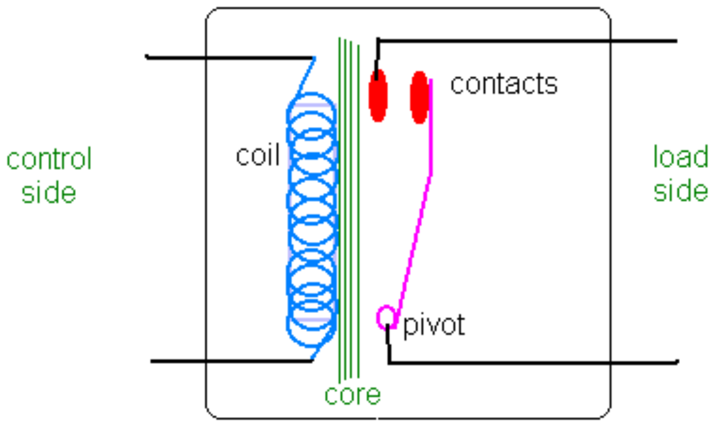


Programmed on/off

ground
V+

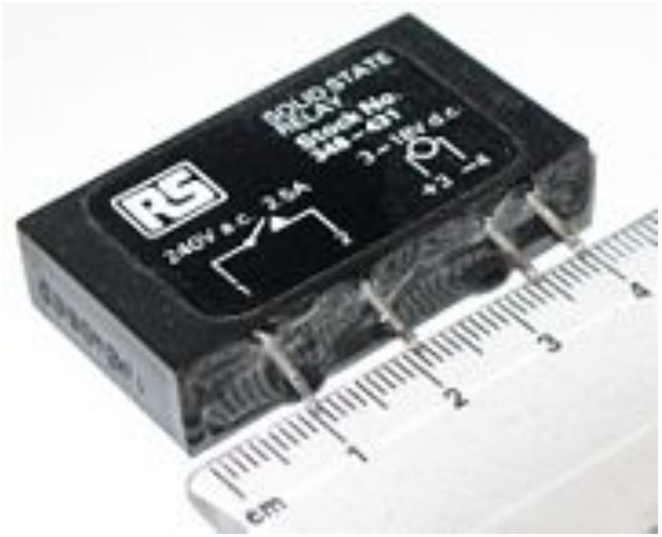

V+
R1
D1
Register Output
R2
Q1
Programmed on/off
V+
ground
1 = LED on
0 = LED off

A relay is an electrical switch that opens and closes under the control of another electrical circuit

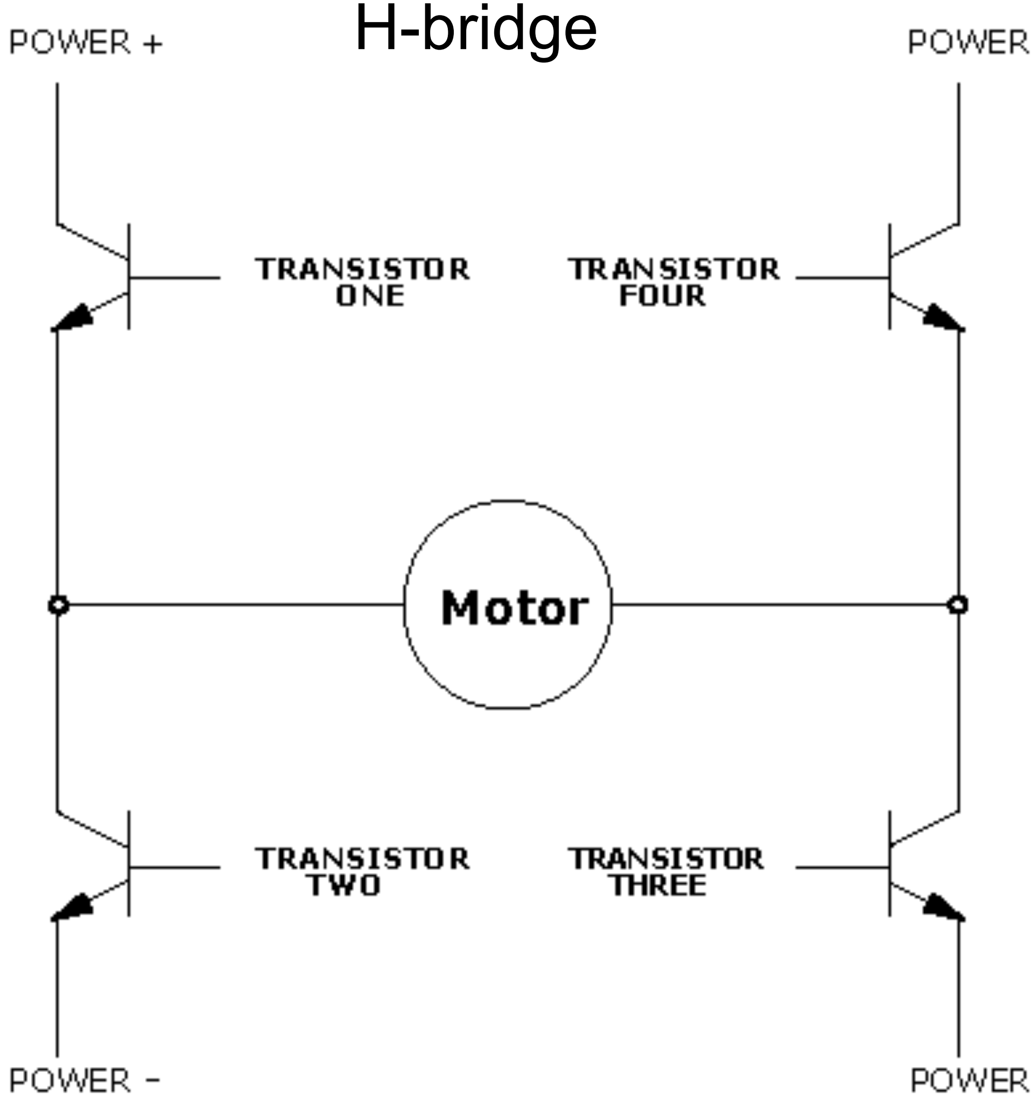


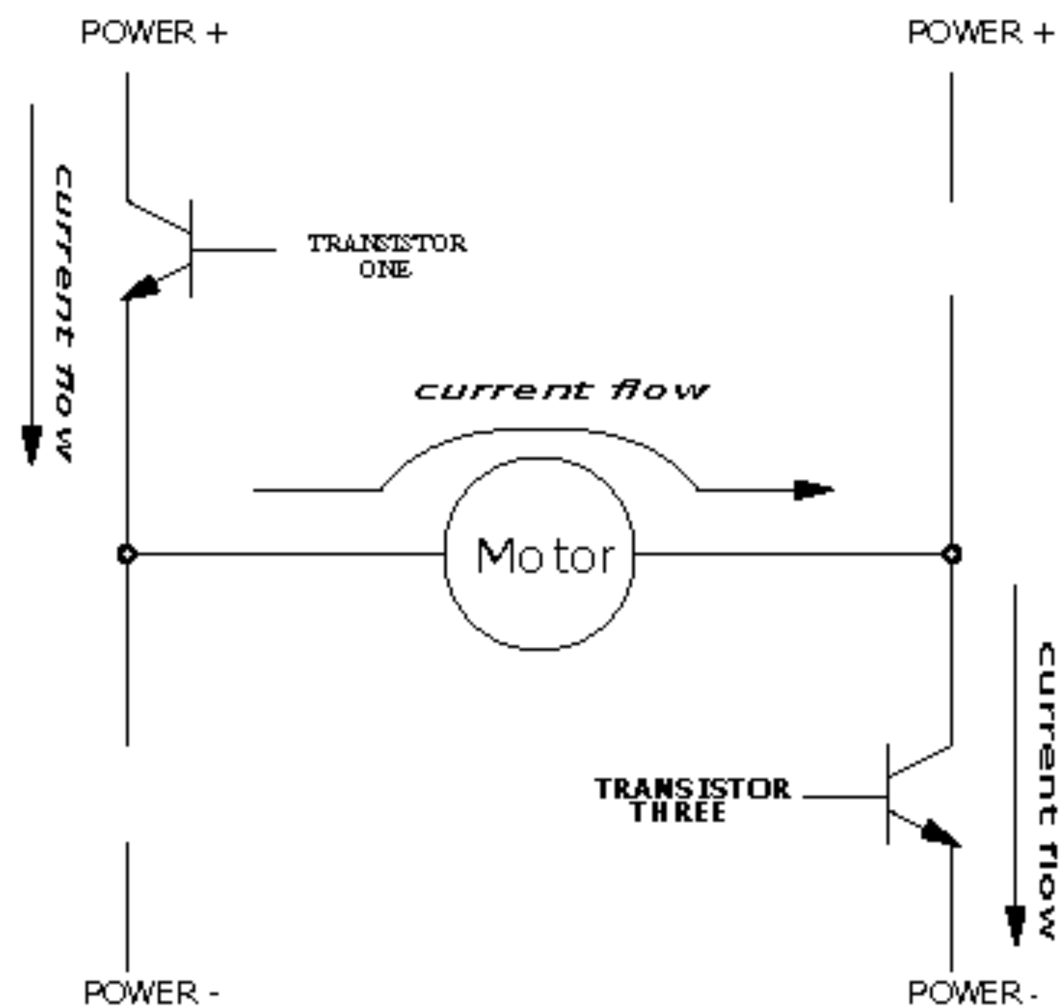
INSIDE A SPST RELAY

pic r-1a



H-bridge


POWER +
POWER
TRANSISTOR ONE
TRANSISTOR FOUR
Motor
TRANSISTOR TWO
TRANSISTOR THREE
POWER -
POWER



POWER +
POWER +
current flow
TRANSISTOR ONE
current flow
Motor
TRANSISTOR THREE
current flow
POWER -
POWER -

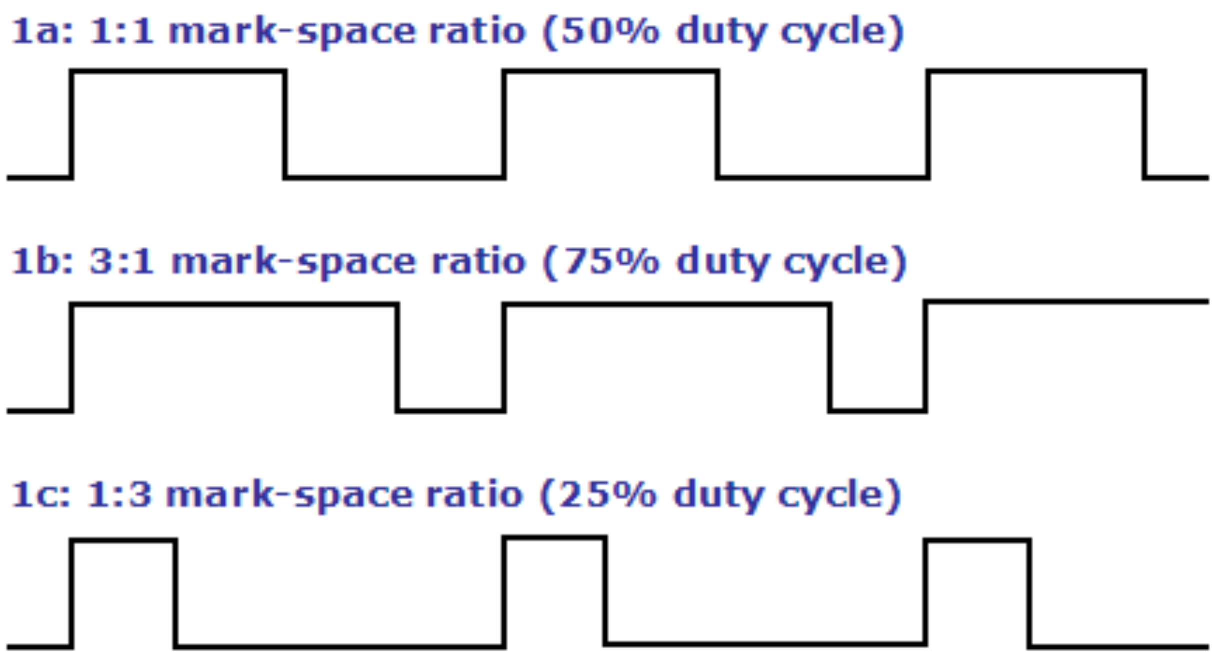


Figure 1: Pulse width modulation waveforms

The speed of the motor is controlled by pulse width modulation (PWM), as shown in the diagram in Figure 3-5. The motor power is rapidly turned on and off over a time interval. The speed of a motor depends on the average voltage applied to it, and the PWM method is a way of controlling this average voltage. It is energy efficient because the transistors are either completely off or on.

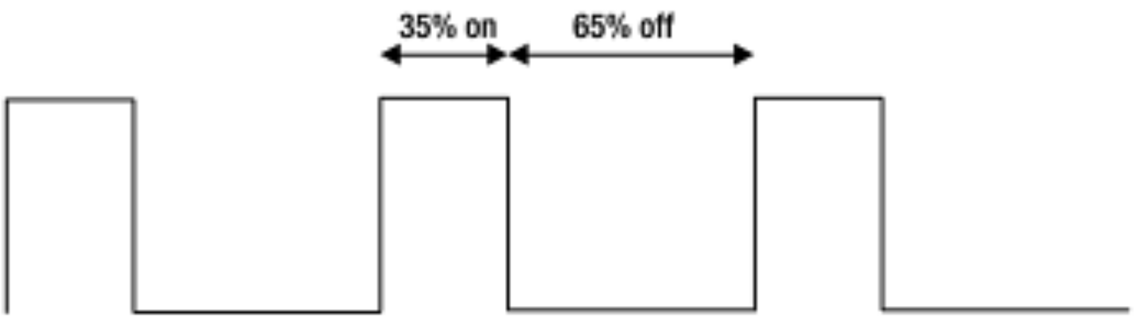


Figure 3-5. *PWM for motor at power level of 35%*

With the standard firmware, the length of the whole cycle is 128μs. This corresponds to 7,800Hz,

voltage = duty cycle * 9 volt

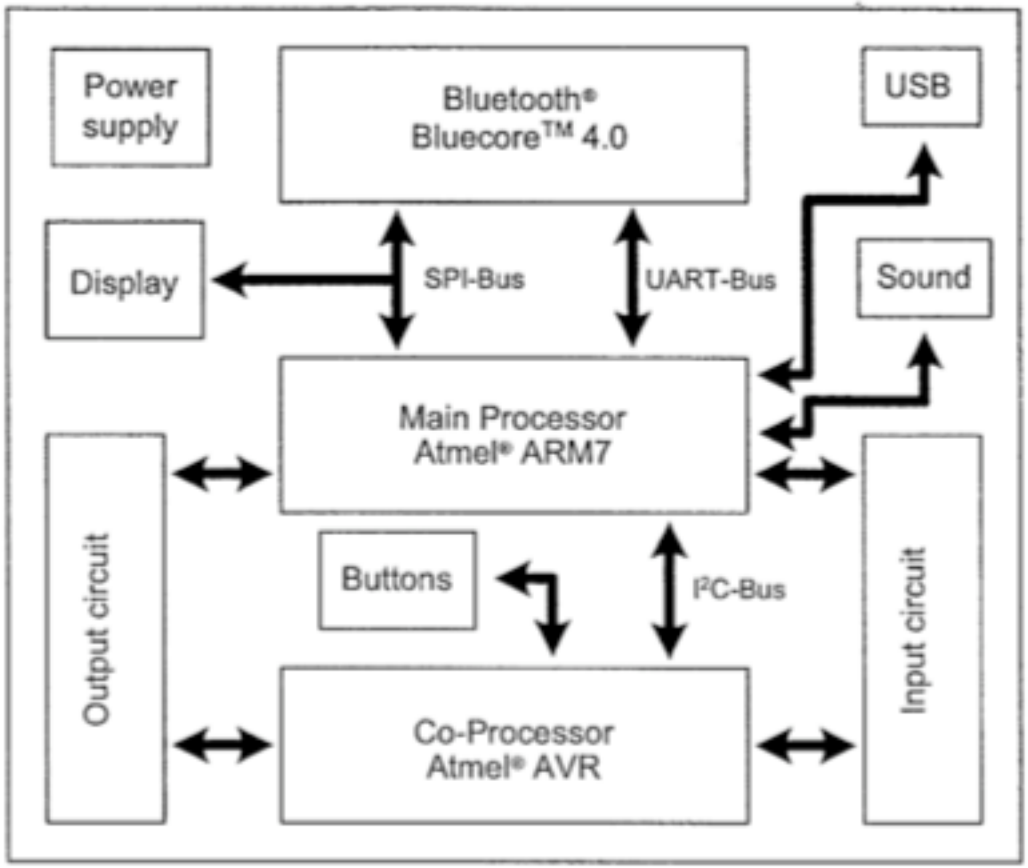
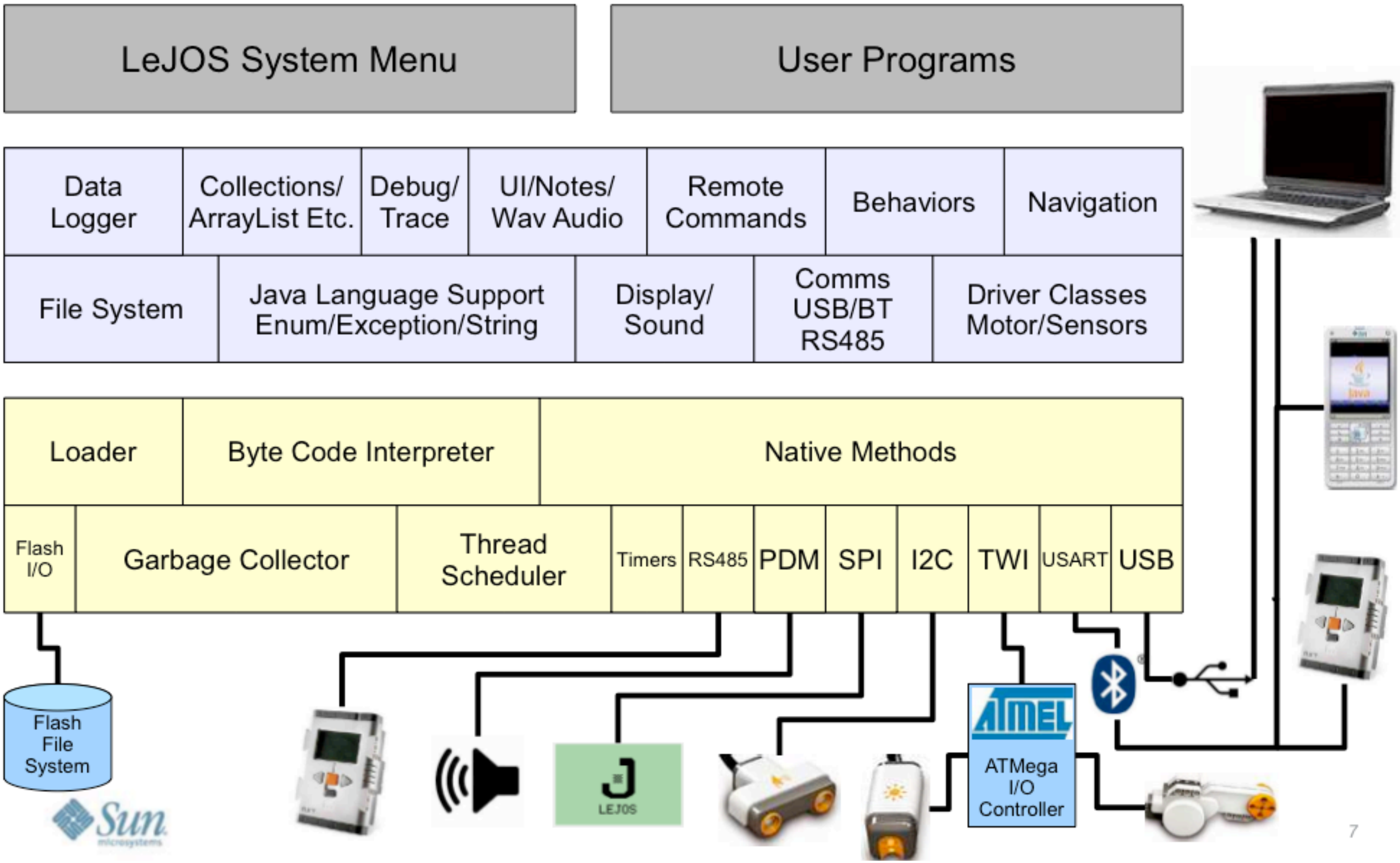
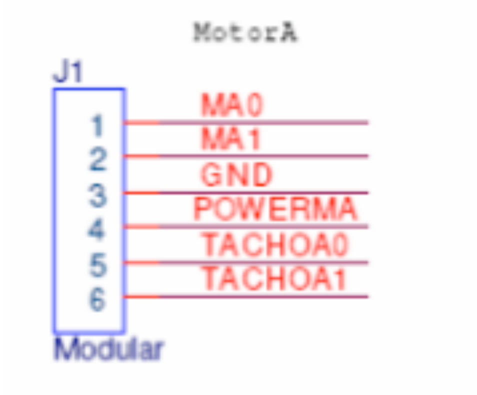


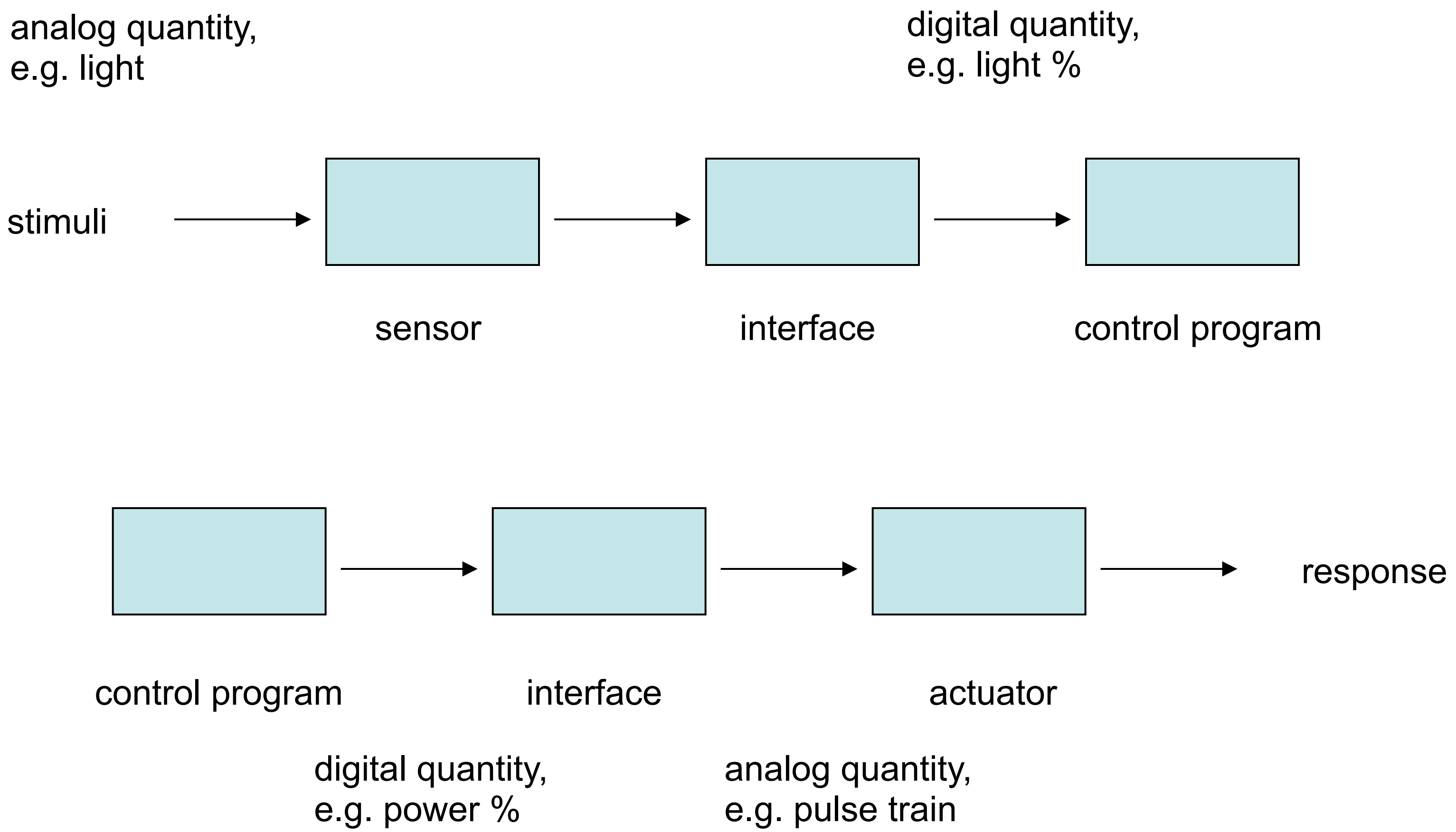
Figure 1: Hardware block diagram of the NXT brick

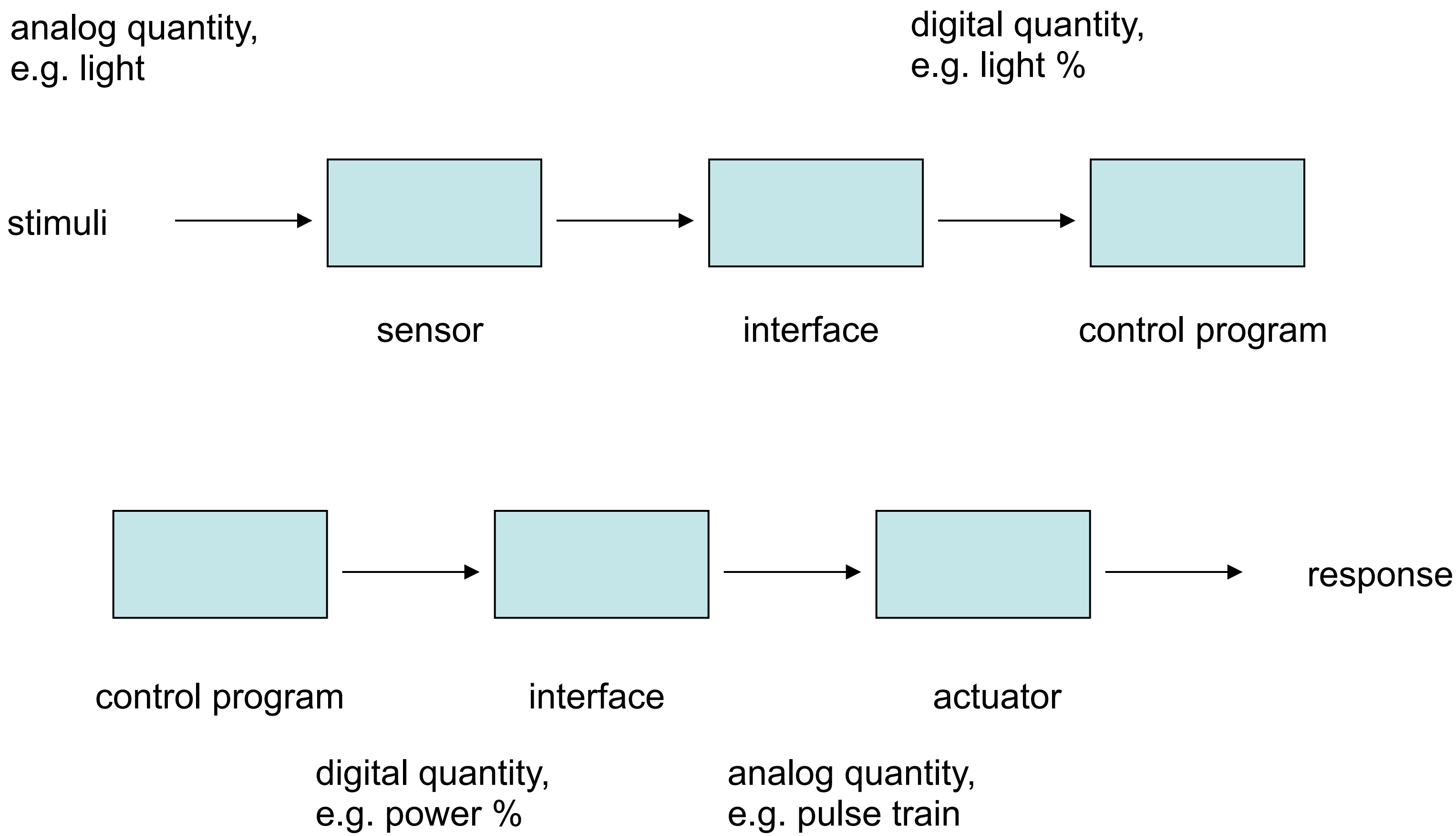
LeJOS Architecture





Pin 1, MA0	PWM output signal for the actuators
Pin 2, MA1	PWM output signal for the actuators
Pin 3, GND	Ground signal related to the output supply
Pin 4, POWERMA	4.3 Volt output supply
Pin 5, TACHOA0	Input value that includes Schmitt trigger functionality
Pin 6, TACHOA1	Input value that includes Schmitt trigger functionality


analog quantity,
e.g. light
digital quantity,
e.g. light %
stimuli
sensor
interface
control program
control program
interface
actuator
response
digital quantity,
e.g. power %
analog quantity,
e.g. pulse train

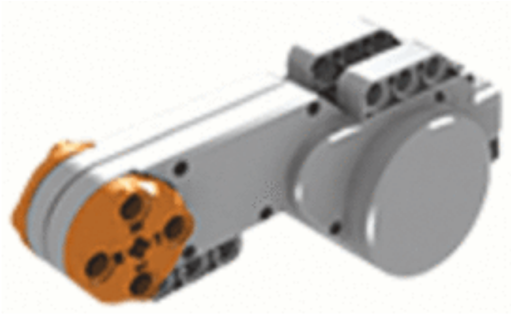


sensor space → actuator space

Embedded Java

The leJOS API classes provide access to the hardware. Especially the output ports and actuators:

MotorPort



Motor

DifferentialPilot

lejos.nxt

Class MotorPort

java.lang.Object
  └ **lejos.nxt.MotorPort**

All Implemented Interfaces:
BasicMotorPort, TachoMotorPort, Encoder

```
public class MotorPort
extends Object
implements TachoMotorPort
```

Abstraction for a NXT output port.

Field Summary

static MotorPort	**A** MotorPort A.
static MotorPort	**B** MotorPort B.
static MotorPort	**C** MotorPort C.

Method Summary

`void`	`controlMotor(int power, int mode)` Low-level method to control a motor.
`int`	`getId()`
`static MotorPort`	`getInstance(int id)` Return the MotorPort with the given Id.
`int`	`getTachoCount()` returns tachometer count
`void`	`resetTachoCount()` resets the tachometer count to 0;
`void`	`setPWMMode(int mode)`

controlMotor

```
public void controlMotor(int power,
                         int mode)
```

Low-level method to control a motor.

Specified by:

`controlMotor` in interface `BasicMotorPort`

Parameters:

`power` - power from 0-100
`mode` - defined in `BasicMotorPort`. 1=forward, 2=backward, 3=stop, 4=float.

```java
package lejos.nxt;

/**
 *
 * Abstraction for a NXT output port.
 *
 */
public class MotorPort implements TachoMotorPort {
	private int _id;
	private int _pwmMode = PWM_FLOAT; // default to float mode

	private MotorPort(int id)
	{
		_id = id;
	}

    /**
     * The number of ports available.
     */
    public static final int NUMBER_OF_PORTS = 3;

	/**
	 * MotorPort A.
	 */
	public static final MotorPort A = new MotorPort (0);

	/**
	 * MotorPort B.
	 */
	public static final MotorPort B = new MotorPort (1);

	/**
	 * MotorPort C.
	 */
	public static final MotorPort C = new MotorPort (2);
```

```
    /**
     * Low-level method to control a motor.
     *
     * @param power power from 0-100
     * @param mode defined in <code>BasicMotorPort</code>. 1=forward, 2=backward, 3=stop, 4=float.
* @see BasicMotorPort#FORWARD
* @see BasicMotorPort#BACKWARD
* @see BasicMotorPort#FLOAT
* @see BasicMotorPort#STOP
     */
    public void controlMotor(int power, int mode)
    {
            // Convert lejos power and mode to NXT power and mode
            controlMotorById(_id,
                                    (mode >= 3 ? 0 : (mode == 2 ? -power: power)) ,
                                    (mode == 3 ? 1 : (mode == 4 ? 0 : _pwmMode)));
    }
```

lejos.nxt

Class Motor

java.lang.Object
  └ **lejos.nxt.Motor**

---

public class **Motor**
extends Object

Motor class contains 3 instances of regulated motors.

Example:

```
Motor.A.setSpeed(720);// 2 RPM
Motor.C.setSpeed(720);
Motor.A.forward();
Motor.C.forward();
Thread.sleep (1000);
Motor.A.stop();
Motor.C.stop();
Motor.A.rotateTo( 360);
Motor.A.rotate(-720,true);
while(Motor.A.isMoving() :Thread.yield();
int angle = Motor.A.getTachoCount(); // should be -360
LCD.drawInt(angle,0,0);
```

lejos.nxt

Class NXTRegulatedMotor

java.lang.Object
  └ **lejos.nxt.NXTRegulatedMotor**

All Implemented Interfaces:
BaseMotor, Encoder, RegulatedMotor, Tachometer

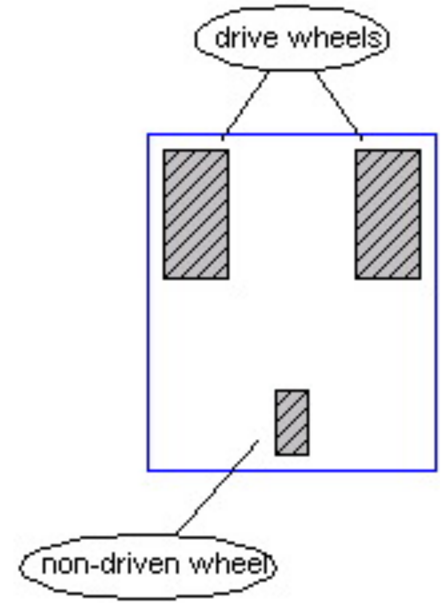
void	`rotate(int angle)` Rotate by the requested number of degrees.
void	`rotate(int angle, boolean immediateReturn)` Rotate by the request number of degrees.
void	`rotateTo(int limitAngle)` Rotate to the target angle.
void	`rotateTo(int limitAngle, boolean immediateReturn)` causes motor to rotate to limitAngle; if immediateReturn is true, method returns immediately and the motor stops by itself and getTachoCount should be within +- 2 degrees if the limit angle If any motor method is called before the limit is reached, the rotation is canceled.

void	`backward()` Causes motor to rotate backwards until `stop()` or `flt()` is called.
void	`flt()` Set the motor into float mode.
void	`flt(boolean immediateReturn)` Set the motor into float mode.
void	`forward()` Causes motor to rotate forward until `stop()` or `flt()` is called.

lejos.robotics.navigation

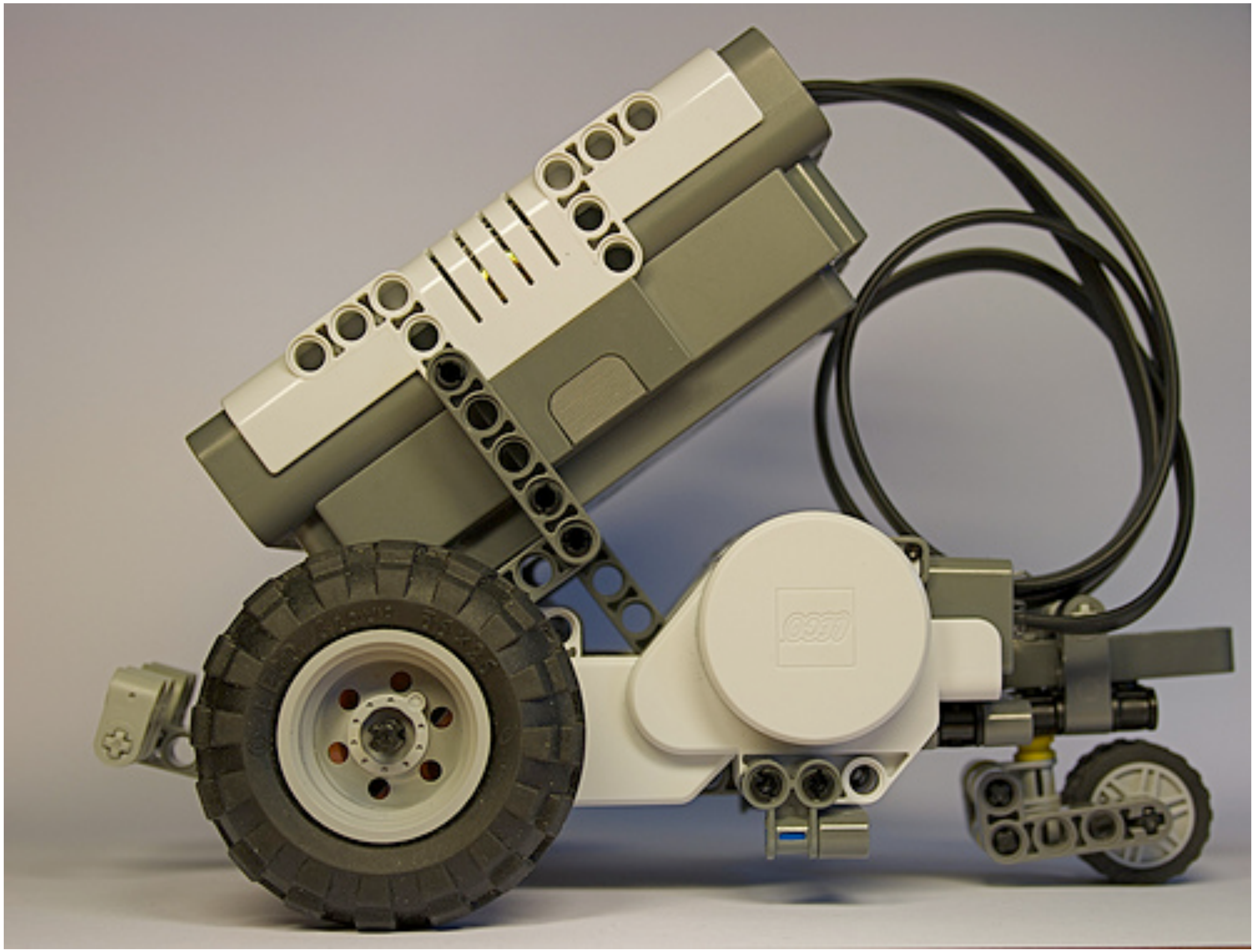
Class DifferentialPilot

java.lang.Object
  └ **lejos.robotics.navigation.DifferentialPilot**



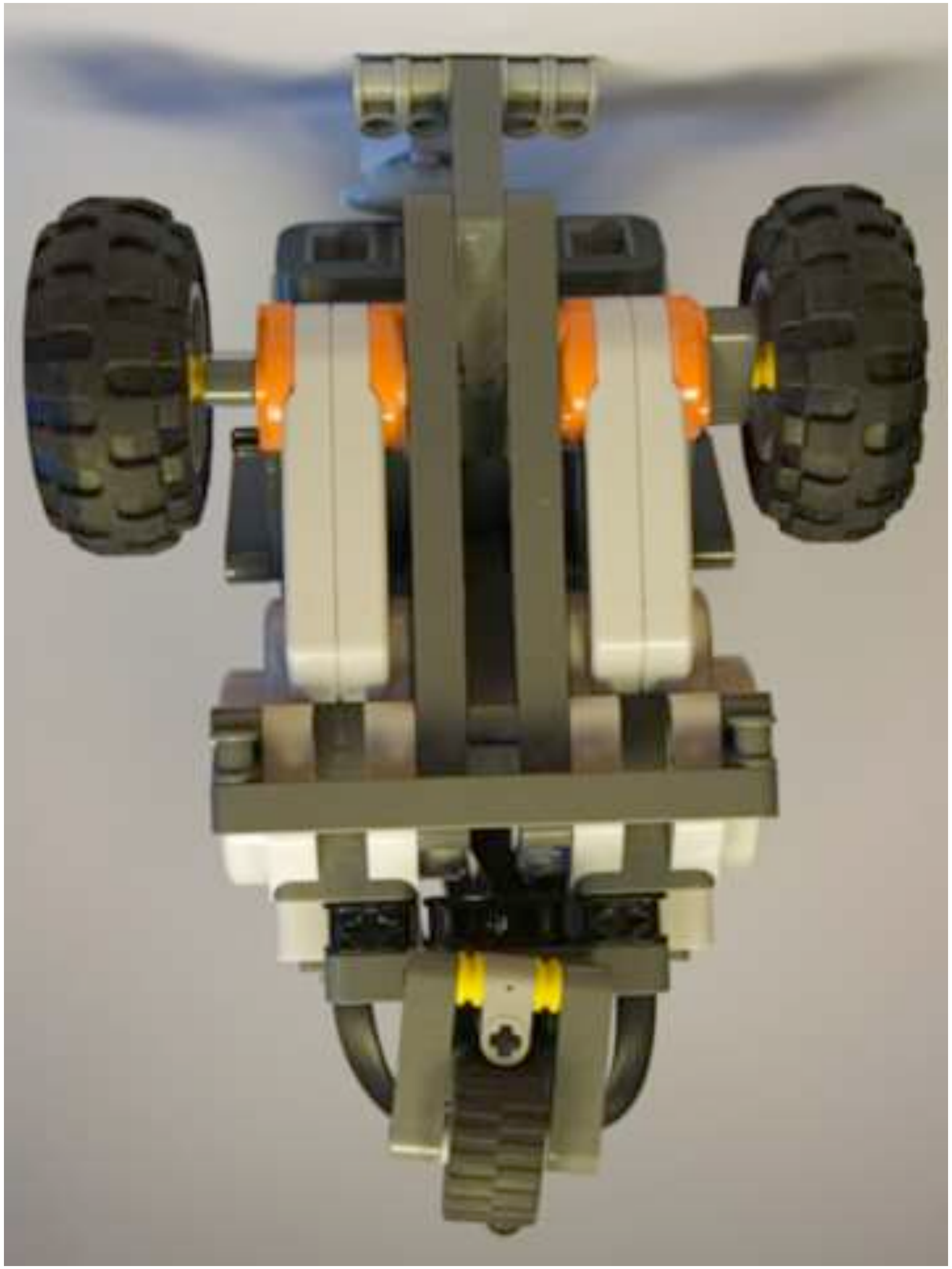

```
DifferentialPilot pilot = new DifferentialPilot(2.1f, 4.4f, Motor.A, Motor.C, true);  // parameters in inches
pilot.setRobotSpeed(30);  // cm per second
pilot.travel(50);         // cm
pilot.rotate(-90);        // degree clockwise
pilot.travel(-50,true);  //  move backward for 50 cm
while(pilot.isMoving())Thread.yield();
pilot.rotate(-90);
pilot.rotateTo(270);
pilot.steer(-50,180,true); // turn 180 degrees to the right
waitComplete();            // returns when previous method is complete
pilot.steer(100);          // turns with left wheel stationary
Delay.msDelay(1000;
pilot.stop();
```

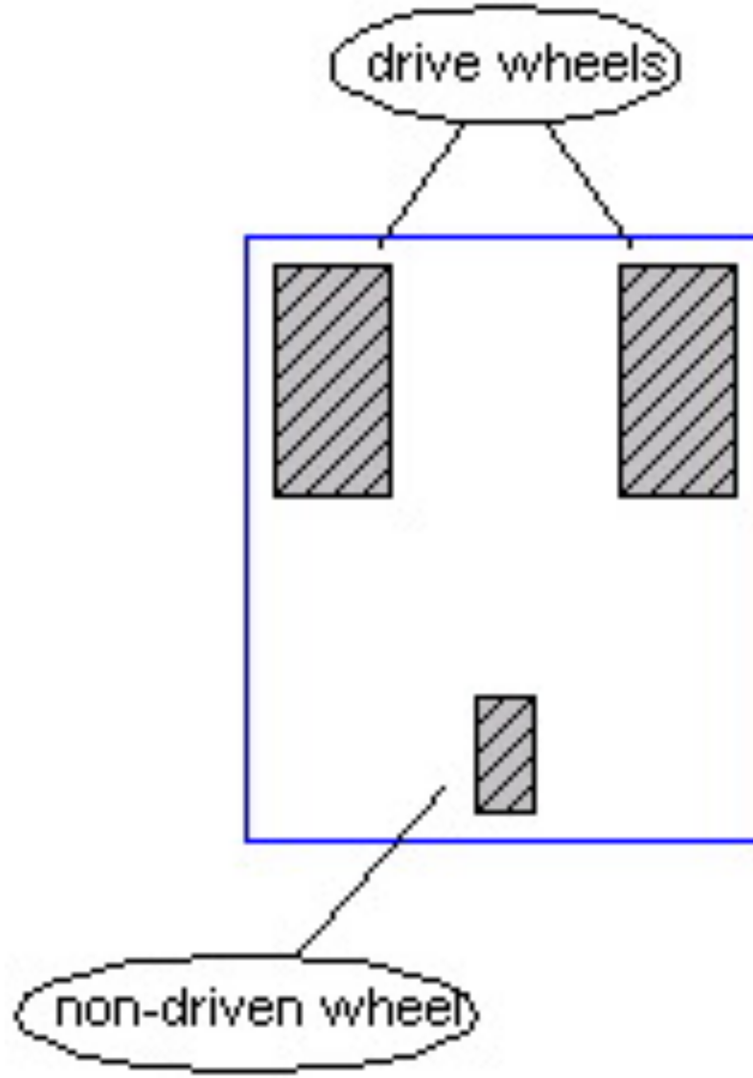
Differential Drive



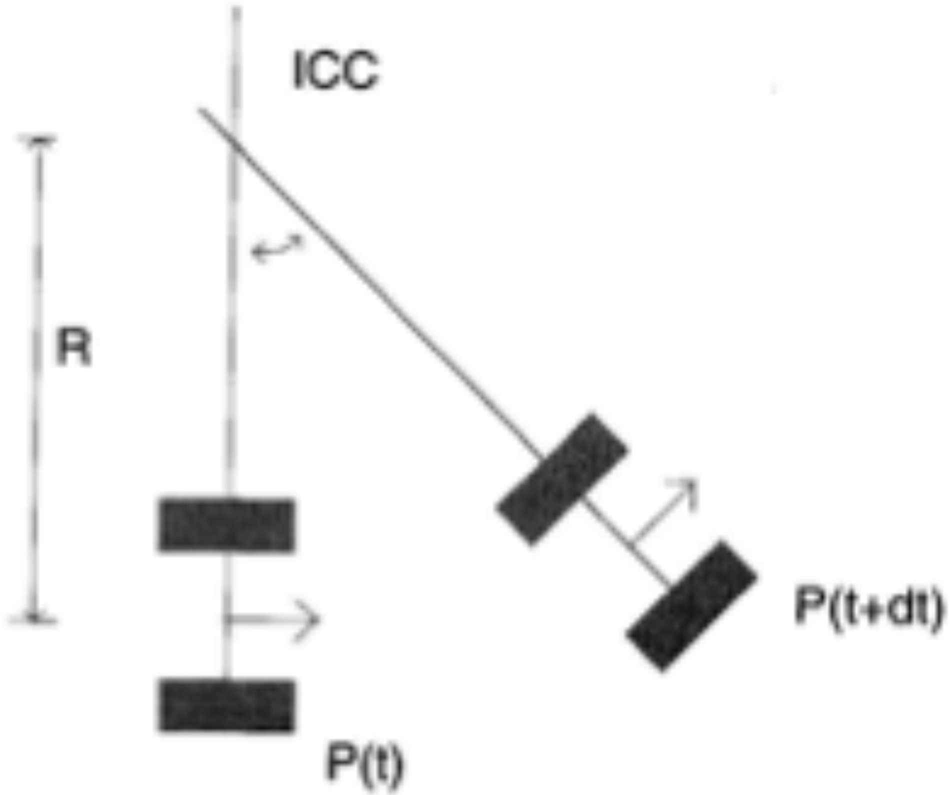
Forward

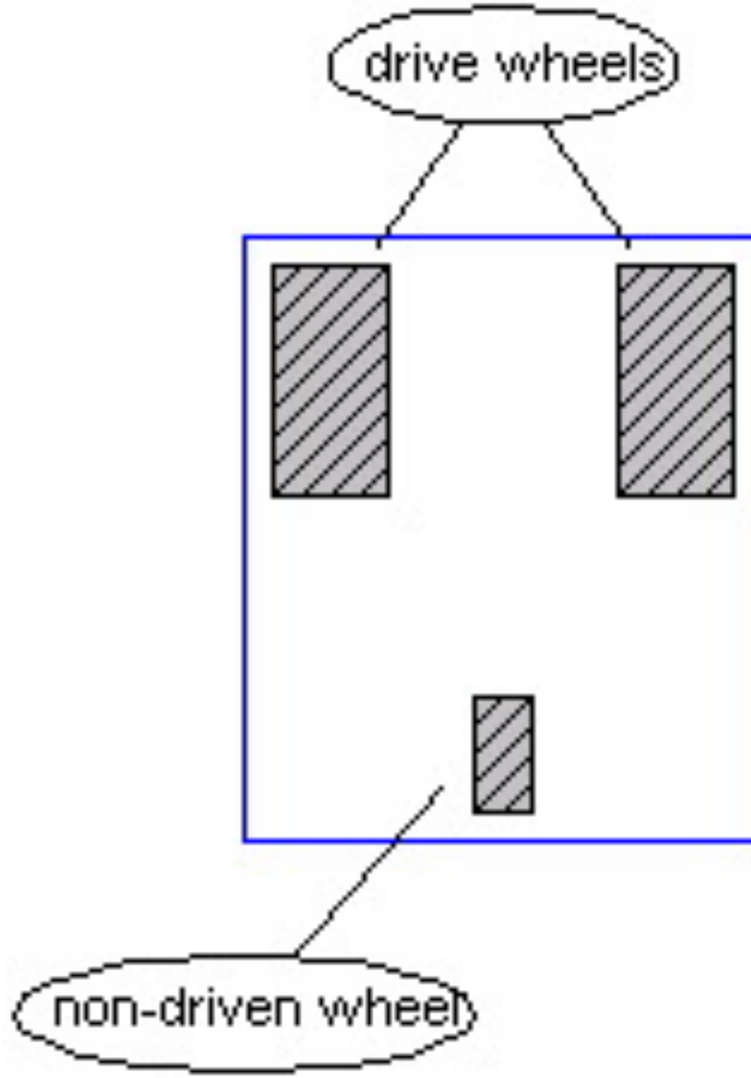
Backward




drive wheels
non-driven wheel

Instantaneous Center of Curvature

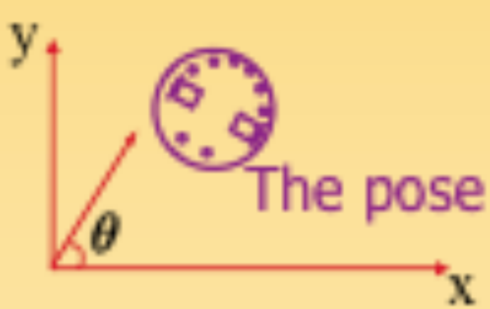




Forward kinematics

Task:

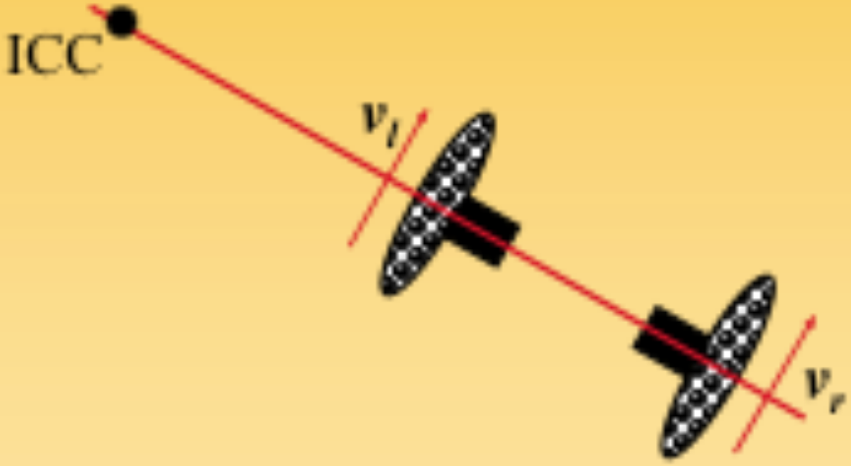
Standing in the pose (x,y,θ) **at time** t,
Determine the pose (x',y',θ') **at time** $t+\delta t$
given the control parameters (v_r, v_l) **!**



The pose (x,y,θ) is defined in a global coordinate system

Differential drive

- Pairs of wheels mounted on a common axis
- If the wheels are rotating on the ground: There is a point ICC !
- By varying (v_r, v_l), ICC moves and different trajectories are chosen

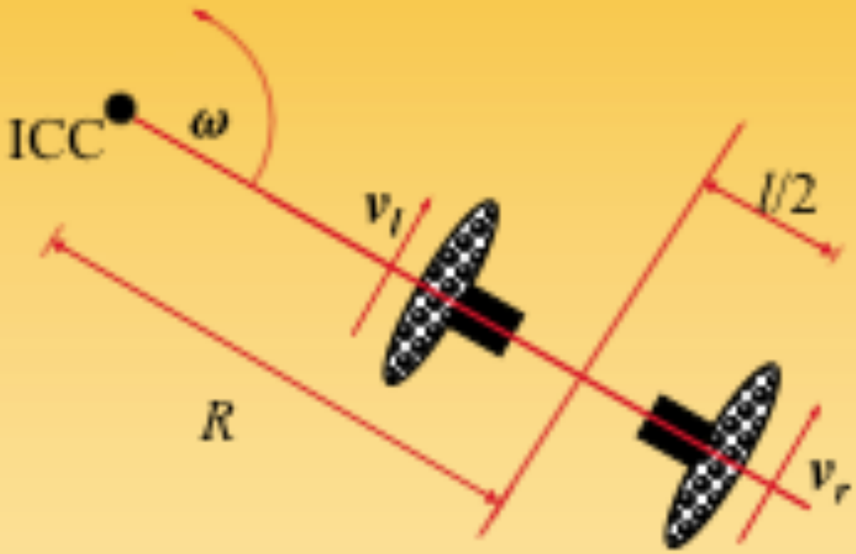


Angular velocity ω

For left and right wheel:

1) $\omega (R + l/2) = v_r$

2) $\omega (R - l/2) = v_l$



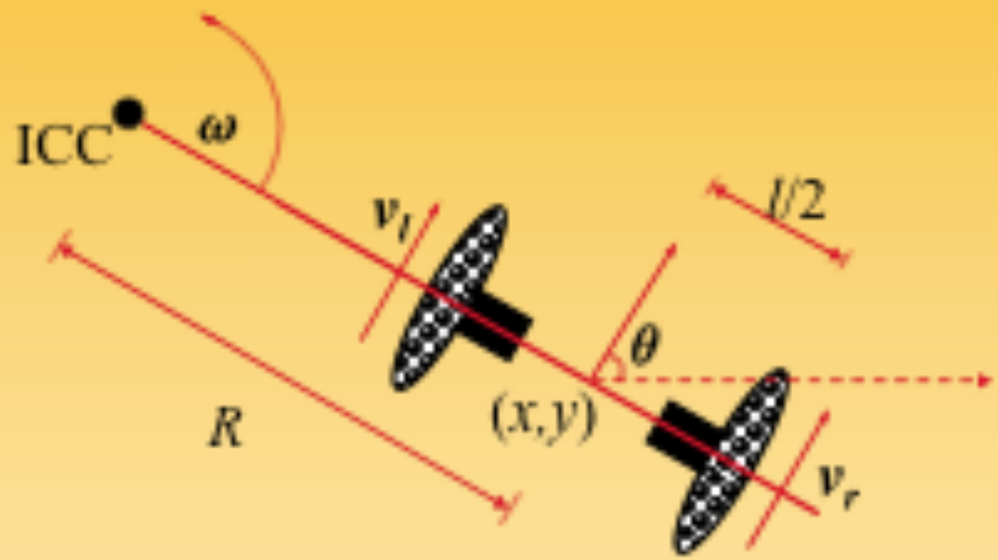
Solve for ω and R :

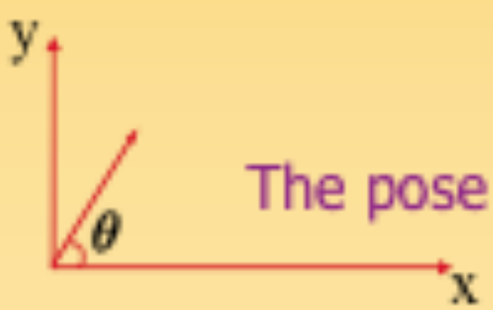
3) $R = l/2(v_l + v_r) / (v_r - v_l)$

4) $\omega = (v_r - v_l) / l$

Forward kinematics

Standing in the pose (x, y, θ) **at time** t,
determine the pose (x', y', θ') **at time** $t + \delta t$
given the control parameters (v_r, v_l) **!**





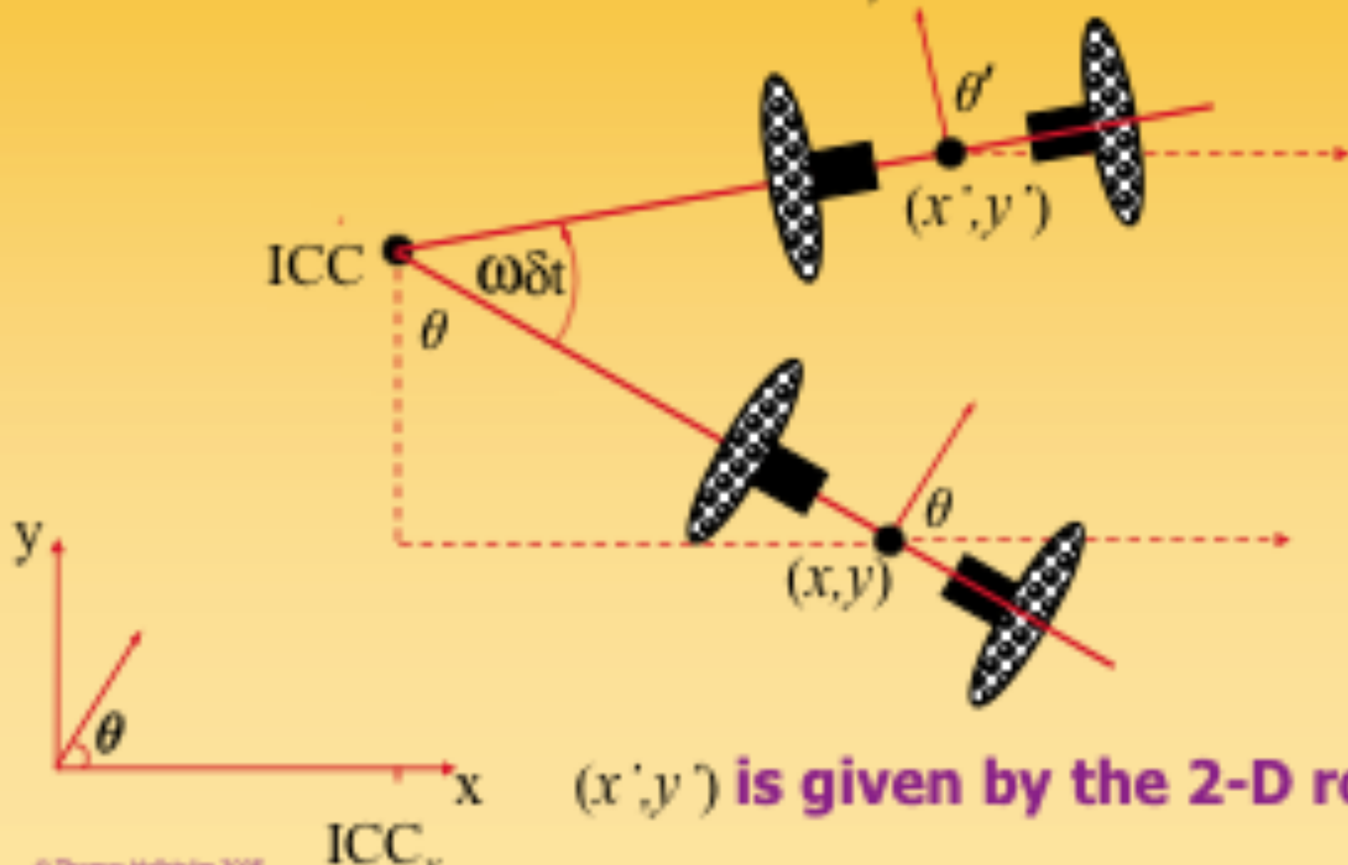
The pose (x, y, θ) is defined in a global coordinate system

Forward kinematics

Rotate around ICC with angular velocity ω for δt seconds:

$$\theta' = \omega\delta t + \theta.$$

$$ICC = [ICC_x, ICC_y] = [\, x - R\sin\theta,\ y + R\cos\theta].$$



(x', y') **is given by the 2-D rotational matrix:**

Forward kinematics

Rotate around ICC with angular velocity ω for δt seconds:

Position at time $t + \delta t$:

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

Wheel encoders give decoder counts $\boldsymbol{n}_r$ and $\boldsymbol{n}_l$ from time=t to $t + \delta t$.
In general: $n\ step = v\ \delta t \Rightarrow v = n\ step/\delta t$
where *step* is the length (mm) of one decoder tick.
Insert in 3) and 4):

$$R = l/2(\boldsymbol{v}_l + \boldsymbol{v}_r) / (\boldsymbol{v}_r - \boldsymbol{v}_l) = l/2(\boldsymbol{n}_l + \boldsymbol{n}_r) / (\boldsymbol{n}_r - \boldsymbol{n}_l)$$

$$\omega\delta t = (\boldsymbol{v}_r - \boldsymbol{v}_l)\ \delta t / l = (\boldsymbol{n}_r - \boldsymbol{n}_l) step\ / l$$

Inverse kinematics

Task:

Standing in the pose (x,y,θ) at time t,
Determine control parameters (v_r, v_l) such that the pose is (x',y',θ') at time $t+\delta t$
Often infinitely many solutions.
Hard to find the optimal solution.

Often easy to find ONE solution by decomposing the problem and controlling only a few DOF at a time

Inverse kinematics
For the Khepera

$v_r = v_l \Rightarrow$

$n_r = n_l \Rightarrow R = \infty, \omega\delta t = 0$

The robot will move in a straight line. I.e.: θ remains the same

$v_r = -v_l \Rightarrow$

$n_r = -n_l \Rightarrow R = 0, \omega\delta t = 2 n_l step / l$

ICC = [ICC_x, ICC_y] = [x, y].

$x' = x, y' = y, \theta' = \theta + \omega\delta t$

The robot will rotate in place about ICC. *I.e.: any θ is reachable.*

(x,y) *remains the same*

NXT Programming

Lesson 3