

Advanced Programming 3

Electronic Toll Collection

Learn About

- How ETC systems work
- Making automatic gates respond to wireless signals

★ Read Programming Basics 1 and 2 and the wireless communications section of Advanced Programming 1 before trying this!

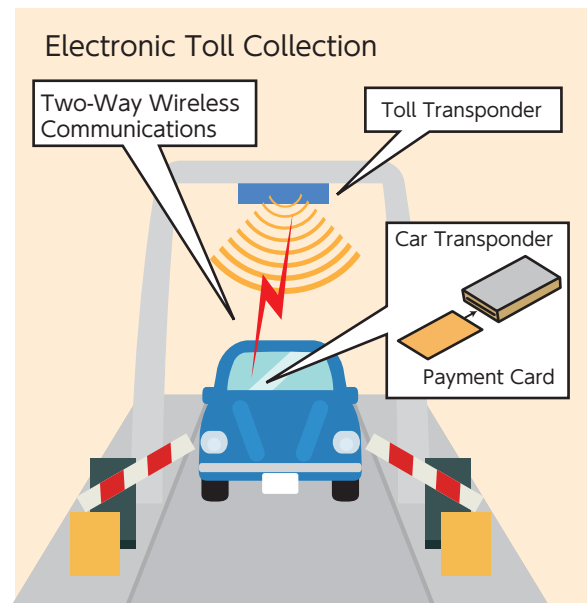
Index

Electronic Toll Collection	2
Appendix	8

Electronic Toll Collection Systems

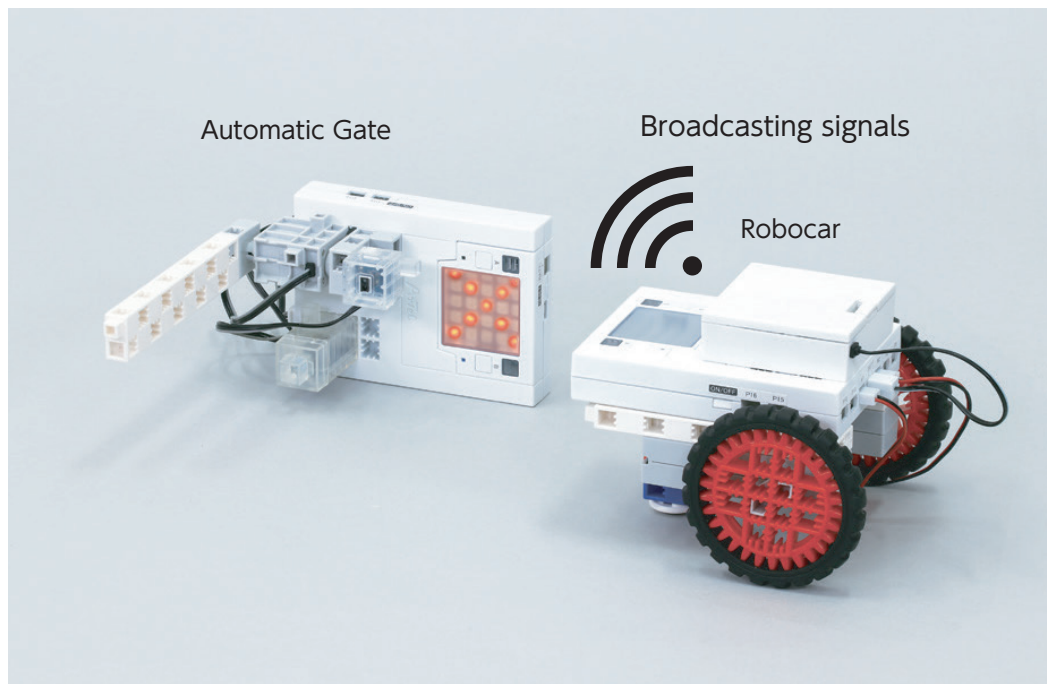
An electronic toll collection system (or ETC system for short) is a kind of highway traffic control system.

These systems help prevent traffic jams by letting cars drive straight through toll gates instead of stopping to pay tolls manually! ETC systems are set up using transponders (devices that can both send and receive data) located both at the toll gate/over the road and inside individual cars, which can wirelessly exchange information with each other to charge and pay the toll.



Your Challenge

Modify a robocar and an automatic gate to make a model ETC system!



1. How the System Works

Let's lay out what conditions we want the ETC system to respond to and how.

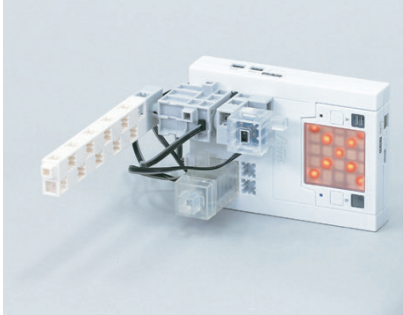
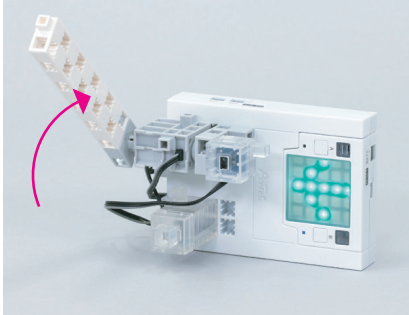
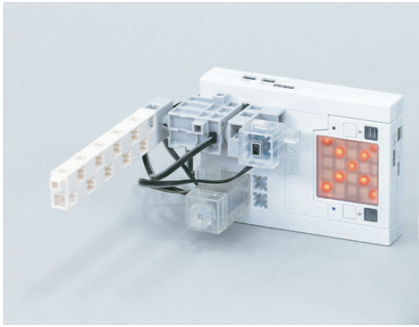
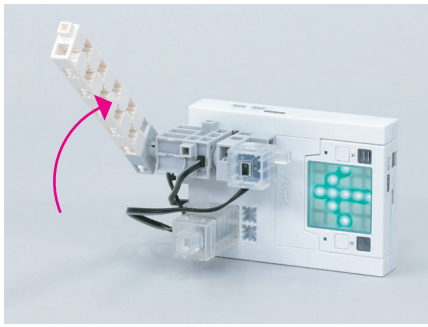
Image	Default State (Waiting)	ETC-Enabled Car Drives Through
	① 	② 
Condition	IR Photoreflexor P1 < Threshold and No signal received from the car	IR Photoreflexor P1 > Threshold and Signal received from the car
What It Does	X appears on the LED display / Gate closes	← appears on the LED display / Gate opens

Image	Car Drives Through Without ETC	Car Pays Toll Manually
	③ 	④ 
Condition	IR Photoreflexor P1 > Threshold and No signal received from the car	Touch Sensor P0 = 0
What It Does	X appears on the LED display / Gate closes	← appears on the LED display / Gate opens

2. Displaying an X

The program below makes an X appear on the LED display.

```
1: from pystubit.board import display, Image
2:
3: img1 = Image('10001:01010:00100:01010:10001:')
4: display.show(img1)
```

Line 1 of this program, **from pystubit.board import display**, imports a **display** object from the Studuino:bit library's **pystubit.board** module.

See Appendix A in Programming Basics 2 to learn more about using **display** objects.

Using **Image** objects lets you create image patterns for the LED display!

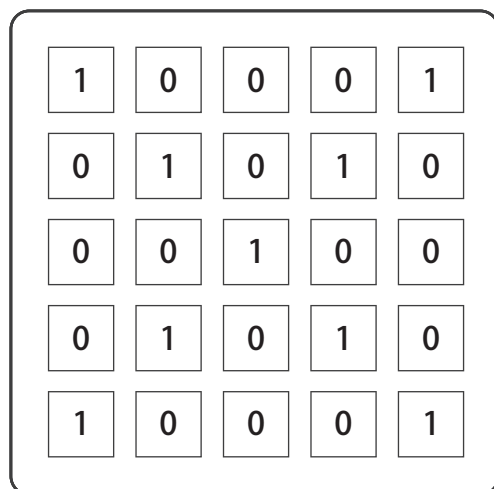
See Appendix A to learn more about using **Image** objects.

Images are made up of color data and a pattern specifying which LEDs in the display should be turned on/off. You can write an image pattern using 0s to mark off LEDs and 1s to mark on LEDs.

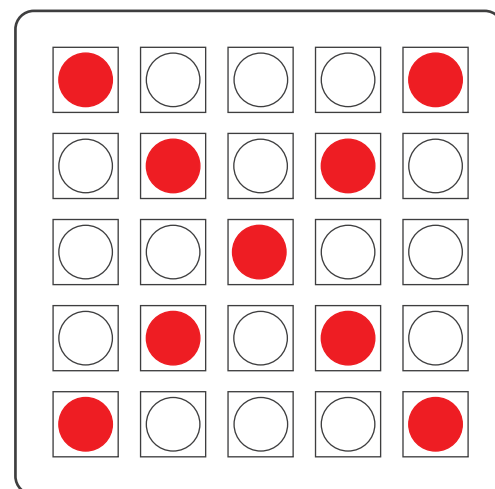
Look at the X image pattern below and what it looks like on the LED display as an example!

The Image

■ Pattern



LED Display



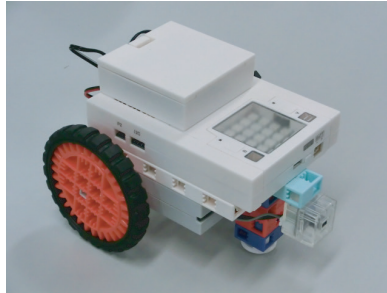
■ Base Color: Red

Once you've made an image pattern, you can use it with a **display** object like on line 4 of the code above to make it appear on the LED display.

3. Example Program

Put your own threshold value in the !

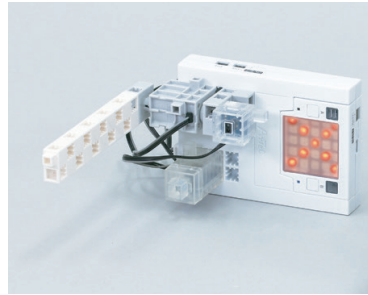
The (Anti-Collision) Robocar



```
1: from pystubit.nw import StuduinoBitRadio
2: from pyatcrobo2.parts import DCMotor,IRPhotoReflector
3: import time
4:
5: car = StuduinoBitRadio()
6: car.on()
7: car.start(0)
8:
9: dcm1 = DCMotor('M1')
10: dcm2 = DCMotor('M2')
11: irp = IRPhotoReflector('P0')
12:
13: dcm1.power(50)
14: dcm2.power(50)
15:
16: while True:
17:     dcm1.ccw()
18:     dcm2.ccw()
19:
20:     if (irp.get_value() >   ):
21:         dcm1.brake()
22:         dcm2.brake()
23:         car.send_string('etc')
24:         while (irp.get_value() >   ):
25:             pass
26:         time.sleep(1)
```

- Line 7 sets the wireless group number to match the automatic gate's group number.
- Lines 9-10 create DC Motor objects, and line 11 creates an IR Photoreflector object.
- Lines 13-14 set the DC Motors' speeds to be slightly slow.
- The code in the **while** statement that starts on line 16 makes up the anti-collision car's driving program.
- Line 23 sends the message **etc** when the car detects a gate ahead and the DC Motors come to a stop.
- Lines 24-25 won't run until the IR Photoreflector doesn't detect a gate ahead anymore.
- Line 26 makes the car wait for a moment after it no longer detects a gate before it starts driving again, to avoid hitting the gate.

The Automatic Gate



```
1: from pystubit.nw import StuduinoBitRadio
2: from pystubit.board import display,Image
3: from pyatcrobo2.parts import Servomotor,TouchSensor,IRPhotoReflector
4: import time
5:
6: gate = StuduinoBitRadio()
7: gate.on()
8: gate.start(0)
9:
10: sv13 = Servomotor('P13')
11: sensor0 = TouchSensor('P0')
12: irp = IRPhotoReflector('P1')
13:
14: img1 = Image('10001:01010:00100:01010:10001:')
15: img2 = Image('00100:00100:10101:01110:00100:')
16: img2.set_base_color((0,20,0))
17:
18: sv13.set_angle(80)
19: display.show(img1)
20:
21: while True:
22:     if sensor0.get_value() == 0:
23:         sv13.set_angle(180)
24:         display.show(img2)
25:         while (irp.get_value() >   ):
26:             pass
27:         time.sleep(3)
28:         sv13.set_angle(80)
29:         display.show(img1)
30:
31:     data = gate.recv()
32:     if not data is None:
33:         if data[1] == 'etc' and (irp.get_value() >   ):
34:             sv13.set_angle(180)
35:             display.show(img2)
36:             while (irp.get_value() >   ):
37:                 pass
38:             time.sleep(3)
39:             sv13.set_angle(80)
40:             display.show(img1)
```

- Line 8 sets the wireless group number to match the car's group number.
- Line 10 creates a Servomotor object, line 11 creates a Touch Sensor object, and line 12 creates an IR Photoreflector object.
- Line 14 creates an X image for the LED display, and line 15 creates a ← image.
- Line 16 changes the color of all the pixels in **img2** to green.
- Line 18 sets the Servomotor's angle to 80 degrees (slightly slanted downward to make it easier for the robocar's IR Photoreflector to detect the gate).
- Line 19 makes an X appear on the LED display.
- Line 22 detects if the Touch Sensor has been pressed and opens the gate and displays a ← if it has been. This represents the toll being paid manually.
- Line 25 won't run until the IR Photoreflector doesn't detect the gate anymore (this means the car has passed through the gate).
- Line 27 makes the car wait for a moment after it no longer detects a gate before it starts driving again, to avoid hitting the gate.
- Line 31 stores received data in a variable called **data**.
- Line 33 runs the same code that runs when the Touch Sensor is pressed, but it's prompted by receiving the message **etc** from the robocar while the IR Photoreflector senses the car in front of the gate. This represents an ETC payment being made.

Appendix: Image Object Methods

Methods	What It Does
<code>__init__(string, color=None)</code> <code>__init__(width=None, height=None, buffer=None, color=None)</code>	Makes image objects using a pattern written in 0s (LED OFF) and 1s (LED ON) in the string parameter. Ex.) <code>Image('01100:10010:11110:10010:10010:',color=(0,0,10))</code> Makes an image object using spaces within the specified width (the width parameter) and height (the height parameter). Ex.) <code>Image(2,2,bytearray[0,1,0,1])</code> Ex.) <code>Image(3,3)</code>
<code>width()</code>	Returns how wide the image is in columns.
<code>height()</code>	Returns how tall the image is in rows.
<code>set_pixel(x,y,value)</code>	The value parameter sets the value of the pixel at coordinates (x, y) in the image. The value parameter can be set to either 0 (OFF) or 1 (ON).
<code>set_pixel_color(x,y,color)</code>	The color parameter sets the color of the pixel at coordinates (x, y) in the image. The color parameter can be set using (R, G, B), [R, G, B] or #RGB.
<code>get_pixel(x,y)</code>	Returns the value (1 or 0) of the pixel at coordinates (x, y) in the image.
<code>get_pixel_color(x,y,hex=False)</code>	Returns the color of the pixel at coordinates (x, y) in the image. If the hex parameter is set to False, the color will be written in (R,G,B). If set to True, it will be written in #RGB.
<code>set_base_color(self,color)</code>	The color parameter sets the color of all the pixels in the image. The color parameter can be set using (R, G, B), [R, G, B] or #RGB.
<code>shift_left(n)</code>	Makes a new image by shifting the current image a number of spaces left set in the n parameter.
<code>shift_right(n)</code>	Identical to <code>shift_left(-n)</code> .
<code>shift_up(n)</code>	Makes a next image by shifting the current image a number of spaces up set in the n parameter.
<code>shift_down(n)</code>	Identical to <code>shift_up(-n)</code> .
<code>copy()</code>	Returns a complete copy of the image.
<code>repr(image)</code>	Get a compact string representing the image.
<code>Str(image)</code>	Get a readable string representing the image.
<code>+</code>	Combine all the pixels from two images to make a new image.

MEMO

