# INTRODUCTION to THE COCKROACH 3000

Rev : January 9, 2006

## Purpose of Document

This document will serve as an introduction to the functionality of the Cockroach 3000. It includes descriptions of how to drive the Cockroach, use its light sensors to detect changes in ambient light levels and read its bumpers to sense hits.

## Cockroach 3000 Control

### Driving Forward

Forward motion is implemented by setting both of the Cockroach's motors to the same speed. The right and left motors are controlled by the RightMtrSpeed and LeftMtrSpeed functions, respectively. The functions require integer arguments between 1 and 10 to move the motors forward.

```
Example: RightMtrSpeed(3)
         LeftMtrSpeed(3)
```

### Driving Backwards

Reverse motion is also implemented by setting the Cockroach motors to the same speed. However, the arguments into the RightMtrSpeed and LeftMtrSpeed functions are required to be integers between –1 and –10.

```
Example: RightMtrSpeed(-5)
         LeftMtrSpeed(-5)
```

### Stopping

The Cockroach is stopped by setting both its motor speeds to 0. This is accomplished by using the RightMtrSpeed and LeftMtrSpeed functions.

### Turning

Turning the Cockroach is accomplished by driving the two motors at different speeds. Depending on the desired effect, there are various ways of turning the Cockroach:

- If both motor speeds are positive, the turn will be gradual
- If one motor is set to a positive speed while the other is set to a negative speed, the turn will be sharp
- Identical but opposite speeds will make the Cockroach spin its current position
- Stopping one motor and driving the other will cause the Cockroach to turn on the stopped wheel.

```
Example: RightMtrSpeed(7)
         LeftMtrSpeed(2)
                 represents a gradual turn to the left.
```

### Reading Changes in Light Level

The amount of light hitting the Cockroach is obtained by using the function LightLevel. The function returns a 10-bit value corresponding to the amount of light seen by the Cockroach's light sensors. A transition by the Cockroach from light to dark, or vice-versa, is sensed by detecting a change in this measured value.

Often, successive values returned by the LightLevel function will vary by a few bits. This causes a problem in implementations that rely on discrete measurements, as in the case of detecting light-to-dark (or dark-to-light) transitions. In order to avoid repeated sensing of a transition due to fluctuating values returned by the LightLevel function, it is beneficial to add a tolerance band (hysterisis) to the transition condition.

Hysterisis may be implemented by running two different threshold tests that depend on two values, LIGHT_THRESHOLD and DARK_THRESHOLD. LIGHT_THRESHOLD defines the minimum light level required for a valid "light" condition and DARK_THRESHOLD defines the maximum light level for a valid "dark" condition. If LIGHT_THRESHOLD and DARK_THRESHOLD are slightly higher and lower than the nominal transition point, respectively, false transitions will be minimized. In the example below a light level of 50 is the mid-point of the hysterisis band.

Example: An event function that tests if the roach has entered the dark may be implemented like this:

```
#define DARK_THRESHOLD = 47;
#define LIGHT_THRESHOLD = 53;

uchar TestIfDark (EVENT_PARAM)
{
  static uchar LastLight = 0;
  static uchar Threshold = DARK_THRESHOLD;

  char GoneDark = if((LightLevel() < Threshold) &&
                     (LastLight >= Threshold));

  if (GoneDark)
    Threshold = LIGHT_THRESHOLD;
  LastLight = LightLevel();
  return (GoneDark);
}
```

The roach's entry into light would then be detected by testing if LightLevel() is greater than Threshold, which is now equal to LIGHT_THRESHOLD. Once the roach is in the light, Threshold is returned to DARK_THRESHOLD, completing the hysterisis implementation.

### Reading Changes in Bumper State

The state of the Cockroach's bumpers is accessed through the functions IsFrontRtBumped(), IsFrontLtBumped(), IsBackRtBumped(), IsBackLtBumped(). These functions return a Boolean value that is true when the bumper is hit and false under normal conditions. These functions use the analog to digital library ADS12 that reads the magnetic field changes for each sensor. These 10-bit magnetic field values can be read using the ReadFrontRt(), ReadFrontLt(), ReadBackRt(), ReadBackLt() functions.