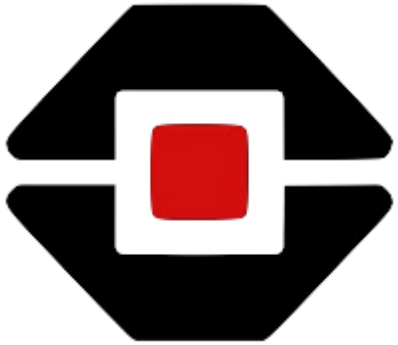


# EV3 Advanced Topics for FLL

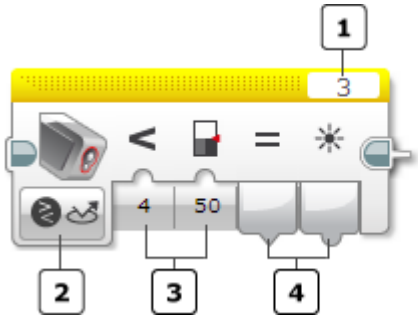
Jim Keller
GRASP Laboratory
University of Pennsylvania
August 2, 2015

# Topics

- Intro to Line Following
  - Basic concepts
  - Calibrate
    - Calibrate the light sensor
    - Display text and data on the Brick
  - Display the Calibration (verify calibration worked as intended)
  - Line Following Techniques (switching and proportional)
    - Switches
    - Loops
- Ultrasonic sensor
  - Measure versus control program flow (i.e. drive until Ultrasonic sensor see something within a specified range)
- How to Use the NXT Light Sensor with EV3
  - Calibration
    - File access block
    - Assigning variables
  - Read raw sensor values
  - Configuring My Blocks

# The EV3 Light Sensor

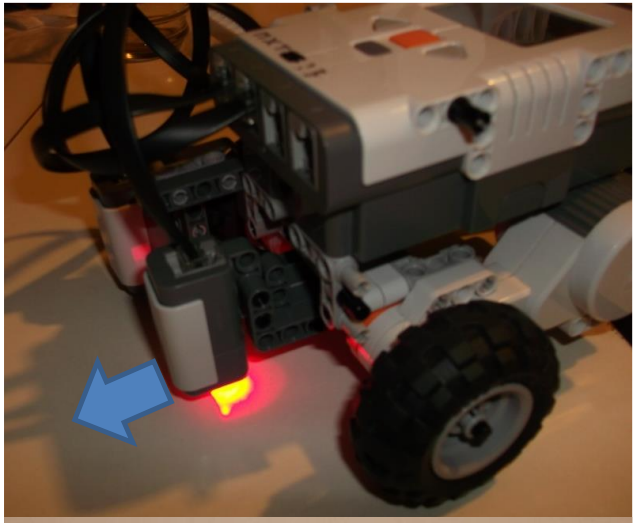

- Senses the brightness of light and converts it to a number between 0 and 100
  - 0 no light detected (darkness)
  - 100 brightest light it can detect
- Has a built-in red light emitting diode (LED) to act like a flashlight
  - Use this feature to prevent shadows and variations in light levels from changing readings. The sensor can be adjusted to take into account conditions
  - our eyes adjust to conditions automatically but for the NXT and EV3 sensors we have to do this manually – this is called ***calibration***




1 Port Selector
2 Mode Selector
3 Inputs
4 Outputs

# What we need to make a robot follow a line

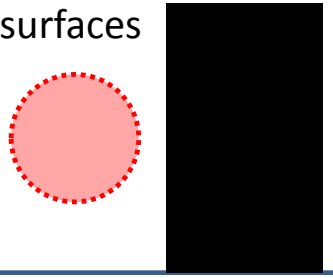
- A light or color sensor facing downward
  - Mount the sensor like this
    - In front of wheels so it is easy to decide how to make it steer
  - Calibrate it for best results (already set for today)
- A wide line (like the ones on the FLL field)
  - the easiest ones to follow don't have any sharp turns or intersections
- A program that will make steering commands to keep the robot over a line



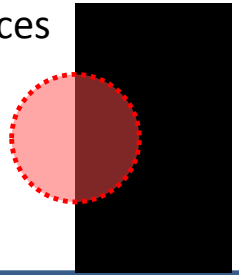
# How the light sensor reacts

- The sensor reading is an ***average*** of the brightness of the entire area it can "see".
  - Mount the sensor on the robot so it is close to the surface on which the wheels roll
    - Some advise it should have a clearance of 2 pennies for best performance
    - As long as it is not much higher than the thickness of the Lego Technic pieces , you'll be OK.
  - If you use the "generate light" option you can see the area the sensor will "see" when you place the robot on a surface.

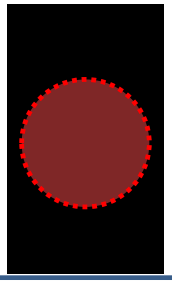
Sensor should read **100** when placed like this if it was calibrated over white and black surfaces



Sensor should read about **50** when placed like this if it was calibrated over white and black surfaces

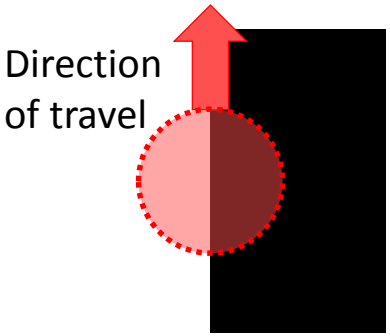


Sensor should read **0** when placed like this if it was calibrated over white and black surfaces

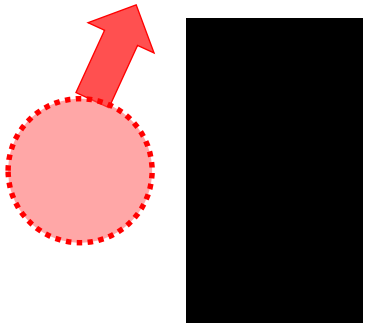


# Line Following Basics

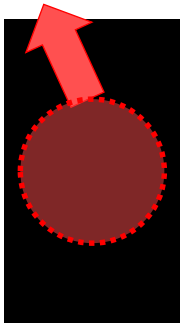
- It is best if the light sensor is slightly in front of the robot's wheels (preferably centered between the wheels but it can be off to one side if necessary)
- The idea is to have the robot follow the ***edge of the line***:




If this is what the robot "sees", we want it to drive straight ahead because it is perfectly over the edge of the line



If this is what the robot "sees", we want it to turn right because it is to the left side of the line (needs to get back to the edge).

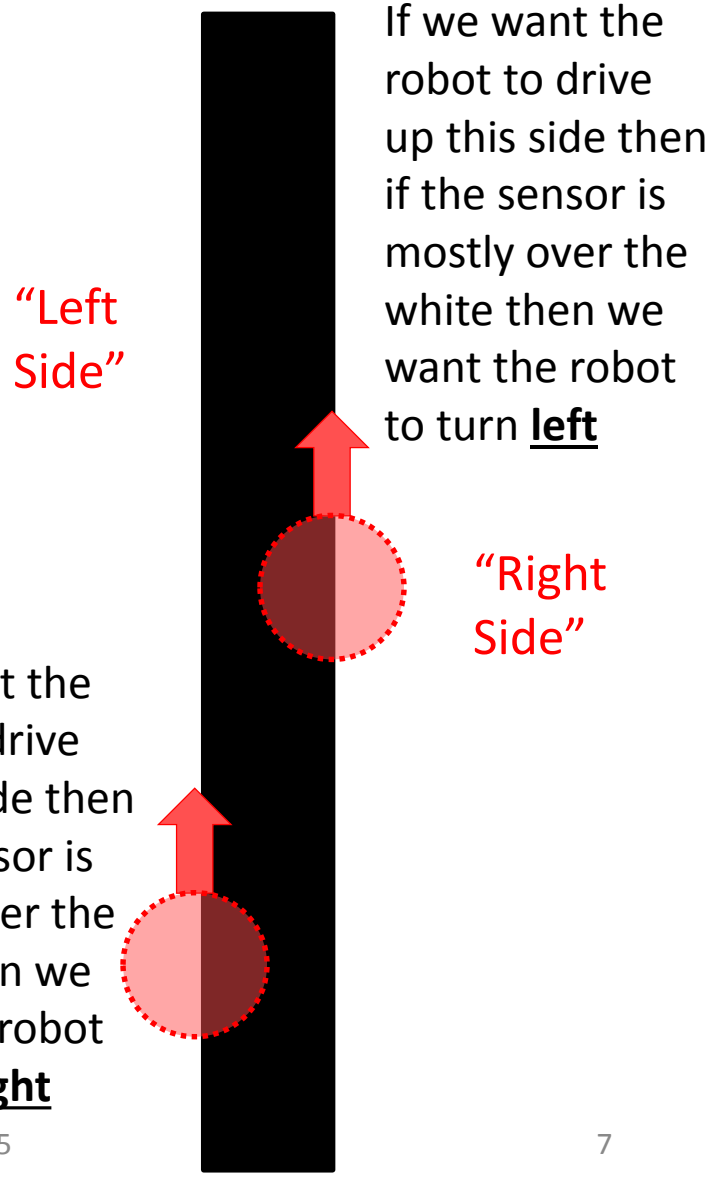


If this is what the robot "sees", we want it to turn left because it is to the right side of the line (needs to get back to the edge).

# What the Program Must Do:

- We need to have the program steer the robot to stay on the edge of the line
  - We can make it follow on either side of the line but we have to choose one side or the other

Notice, once we decide which side the robot is going to use, if it ever finds itself on the other side, it will make the ***wrong*** decision, so we have to be sure it can steer quickly enough to never pass over the line completely

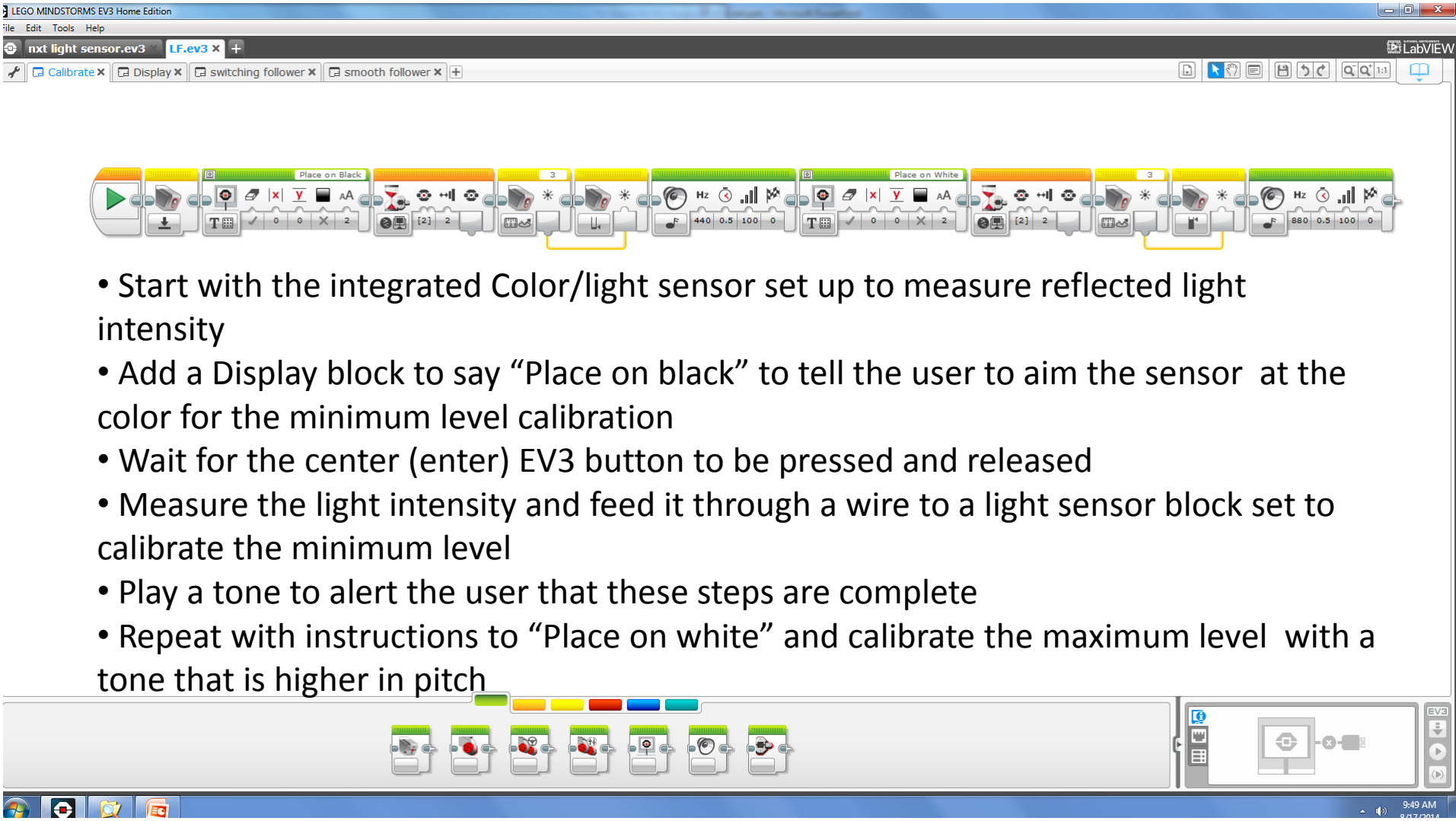


# What Blocks Do We Need

- A Loop (to keep repeating steering commands to follow the line until we want to stop)
- A Light sensor (to sense where the robot is)
- *A block or group of blocks to choose which direction to steer – this is where the "magic" happens.*
- A Move/Steering block (to move the robot)

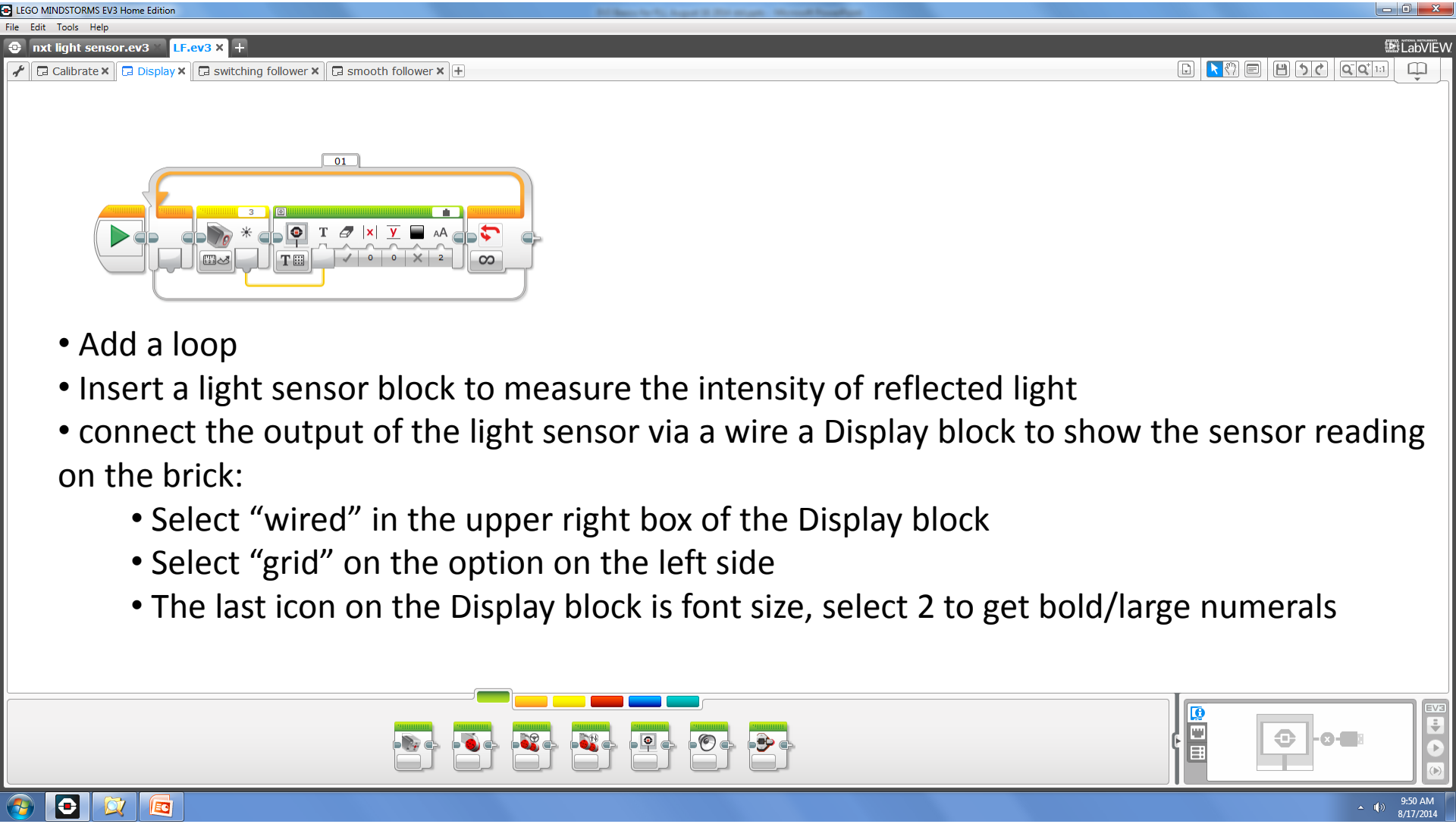
Before programming the line follower, we need to be sure the light sensor is properly calibrated

# Calibration of the EV3 Color/Light Sensor for Reflected Light Intensity



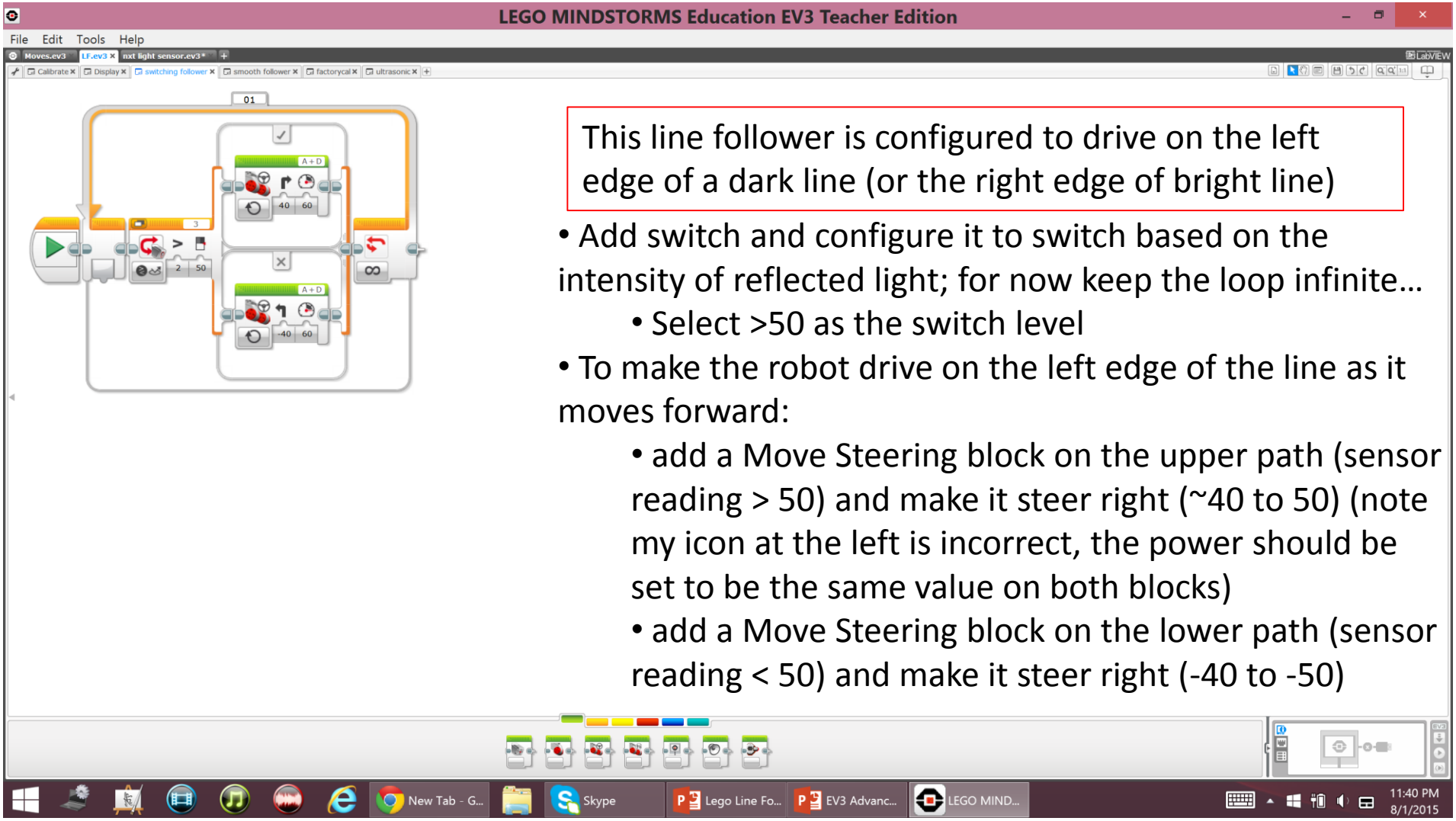

- Start with the integrated Color/light sensor set up to measure reflected light intensity
- Add a Display block to say “Place on black” to tell the user to aim the sensor at the color for the minimum level calibration
- Wait for the center (enter) EV3 button to be pressed and released
- Measure the light intensity and feed it through a wire to a light sensor block set to calibrate the minimum level
- Play a tone to alert the user that these steps are complete
- Repeat with instructions to “Place on white” and calibrate the maximum level with a tone that is higher in pitch

# How to Display/Verify a Calibration




- Add a loop
- Insert a light sensor block to measure the intensity of reflected light
- connect the output of the light sensor via a wire a Display block to show the sensor reading on the brick:
    - Select “wired” in the upper right box of the Display block
    - Select “grid” on the option on the left side
    - The last icon on the Display block is font size, select 2 to get bold/large numerals

# A Simple Switching Line Follower




This line follower is configured to drive on the left edge of a dark line (or the right edge of bright line)

- Add switch and configure it to switch based on the intensity of reflected light; for now keep the loop infinite…
  - Select >50 as the switch level
- To make the robot drive on the left edge of the line as it moves forward:
  - add a Move Steering block on the upper path (sensor reading > 50) and make it steer right (~40 to 50) (note my icon at the left is incorrect, the power should be set to be the same value on both blocks)
  - add a Move Steering block on the lower path (sensor reading < 50) and make it steer right (-40 to -50)

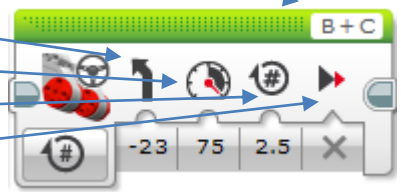
# How a Little Math Can Make a Smooth Line Follower Easy to Set up and Tune

- We'll need to use the following concepts that are all within your students' grasp:
  - Subtraction
  - Multiplication
  - Negative numbers

- Engineers use Math to represent ideas so they are easy to understand by others and do the same thing every time. We're going to do the same thing here.

# A Close Look at the Move Steering Block

- So far, we've been manually setting up the Move block by filling in all of the following options each time we use one:
  - Port
  - Steering
  - Power (speed)
  - Duration
  - Next Action



# Steering from Within the Program

- We only need to worry about the steering specs from the previous table:

| | | | | | |
|---|---|---|---|---|---|
|  | Steering | Number | -100 - 100 | < 0 = Steer towards left motor,<br>> 0 = Steer towards right motor | |

  - Commands less than zero steer left
  - Commands greater than zero steer right
  - Zero means drive straight

  - Now you can see that we are going to need to use negative numbers… And why we are going to follow the left edge (if we drive on the left edge of a dark line then if we subtract 50 from the light sensor reading we get negative numbers when we should turn left and positive numbers when we should turn right!).

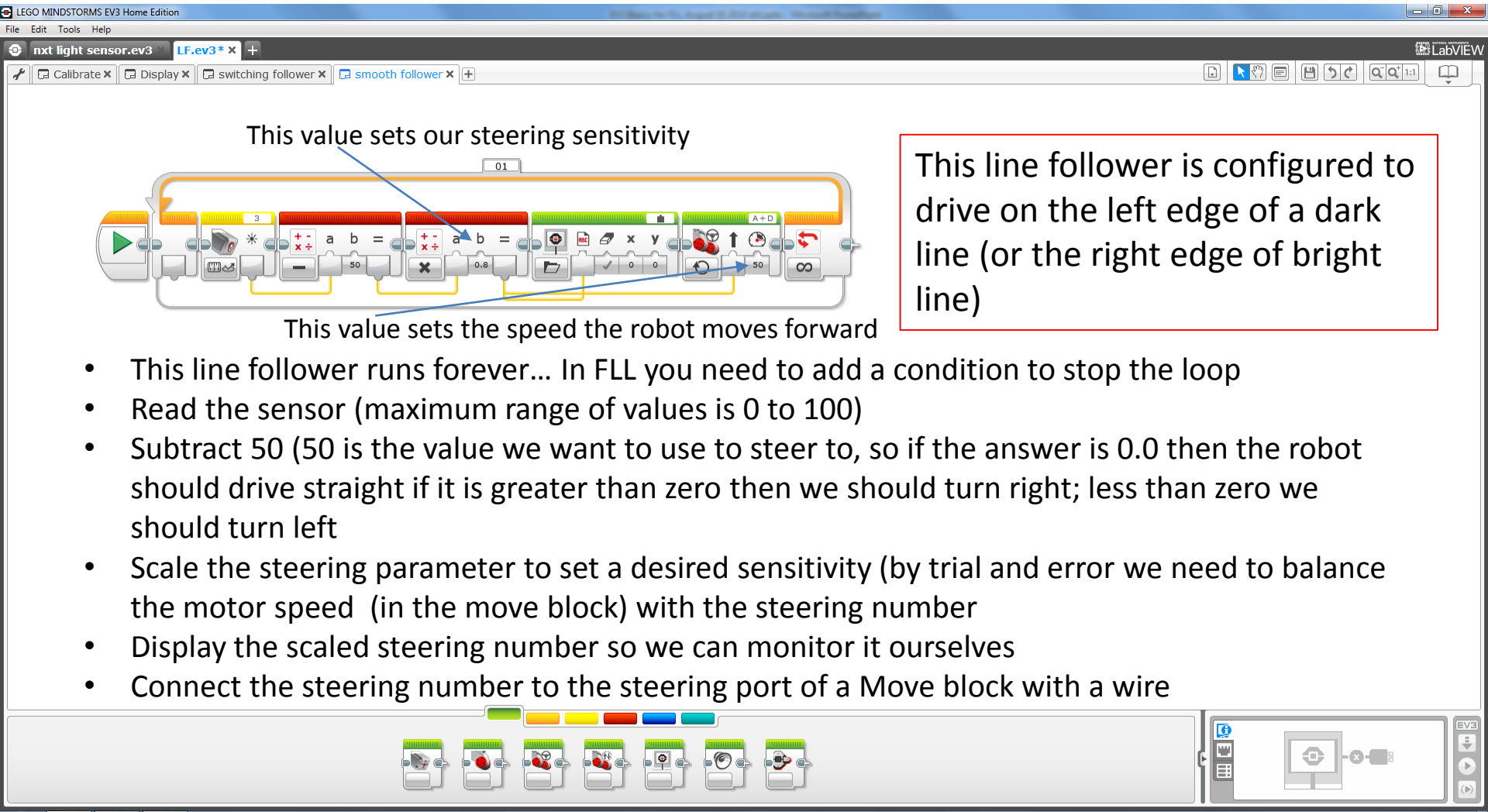
# Steering: the Big Picture

- Our prior programs could only steer at fixed values… in order to get smooth performance we will need many steering values (based on how far from the edge the robot is).
- When we ride a bicycle, the amount we turn the handle bar is determined by how hard we want to turn. We'll have the robot use the light sensor to indicate how hard we should steer or turn its "handle bar"… the steering command
  - The further the light sensor reading is away from 50, the harder we should turn… That is why engineers call this type of controller a Proportional controller because the size of the corrections it makes are proportional to the size of the error (not fixed like our earlier switching controllers).

# How Are We Going to Set up a Steering Command ***Inside*** the Program?

- Look back at the way the light sensor works:
  - It has the range 0 to 100 (0 means dark, 100 means bright)
  - We know we want to steer to the left when the sensor reports a number less than 50 and we want to steer to the right when the sensor reports a number greater than 50…
  - So let's subtract 50 from the light sensor output
    - Now the result of this operation is:
      - ***negative*** if the reading is less than 50 (in the extreme the result is -50 if the sensor reading is 0 because 0 – 50 = -50)
      - ***positive*** if the reading is greater than 50 (in the extreme the result is +50 if the sensor reading is 100 because 100 – 50 = +50)
      - We have something that looks like the steering command it just has half the range…
      - The only thing that remains is to scale it. Let's see what this looks like…

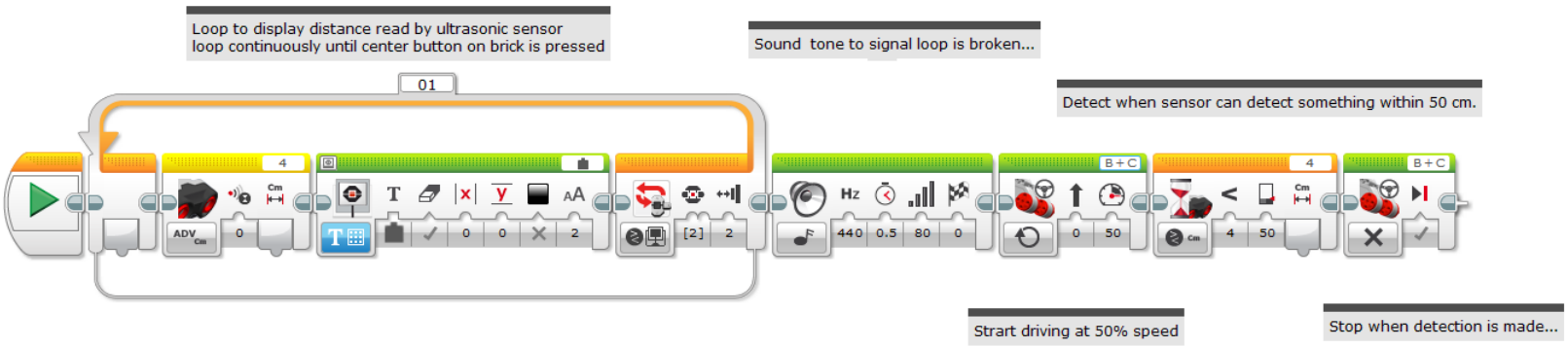
# A Smooth (proportional) Line Follower




This line follower is configured to drive on the left edge of a dark line (or the right edge of bright line)

- This line follower runs forever… In FLL you need to add a condition to stop the loop
- Read the sensor (maximum range of values is 0 to 100)
- Subtract 50 (50 is the value we want to use to steer to, so if the answer is 0.0 then the robot should drive straight if it is greater than zero then we should turn right; less than zero we should turn left
- Scale the steering parameter to set a desired sensitivity (by trial and error we need to balance the motor speed (in the move block) with the steering number
- Display the scaled steering number so we can monitor it ourselves
- Connect the steering number to the steering port of a Move block with a wire

# Ultrasonic Sensor tips

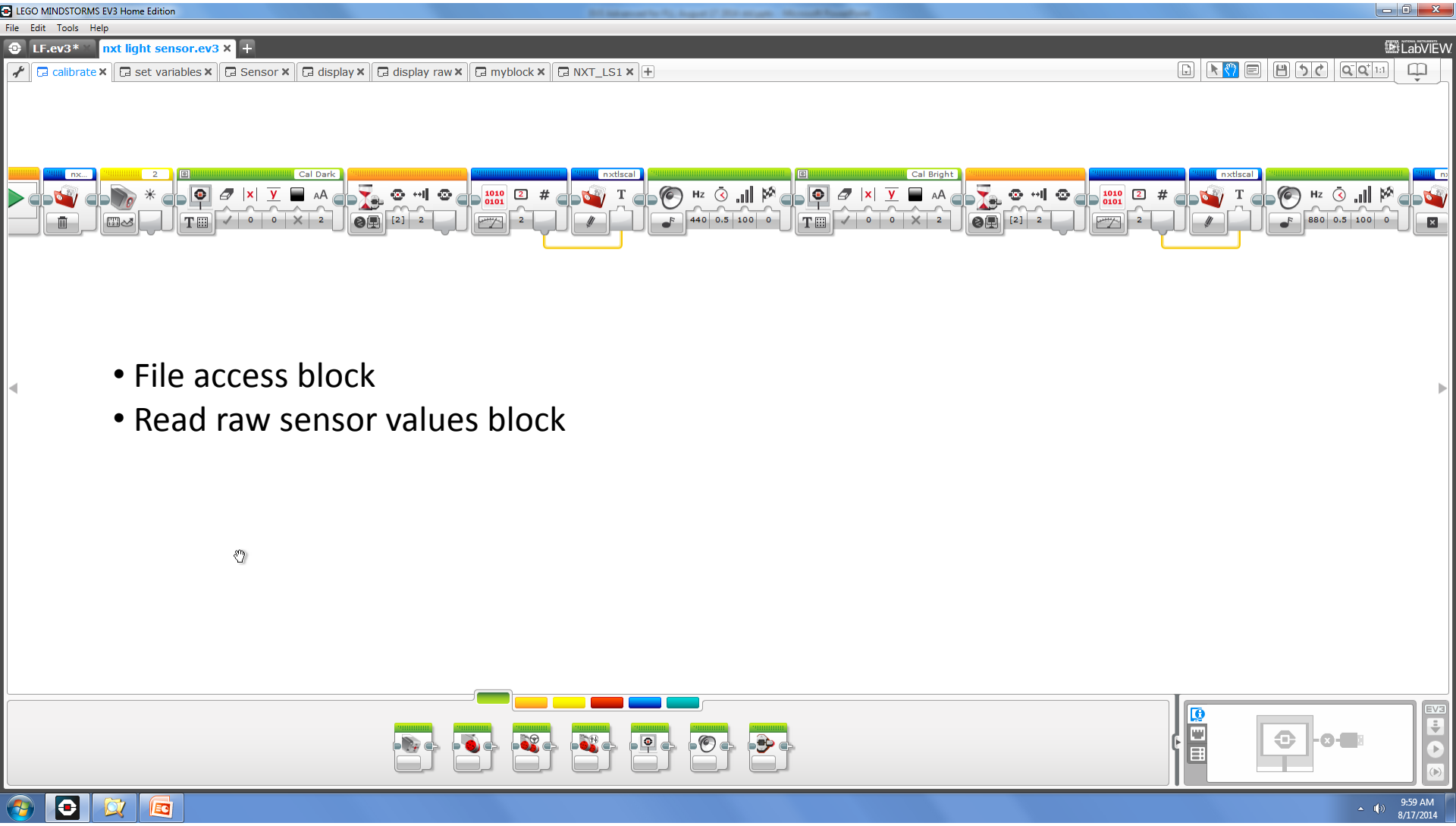



- Loop simply displays value of distance detected by Ultrasonic sensor
  - Stop loop by pressing center button on the brick
  - Play a tone to signal the loop has been stopped
- Move robot until it senses an obstacle 50 cm in front of it

# Integrating the NXT Light Sensor with EV3

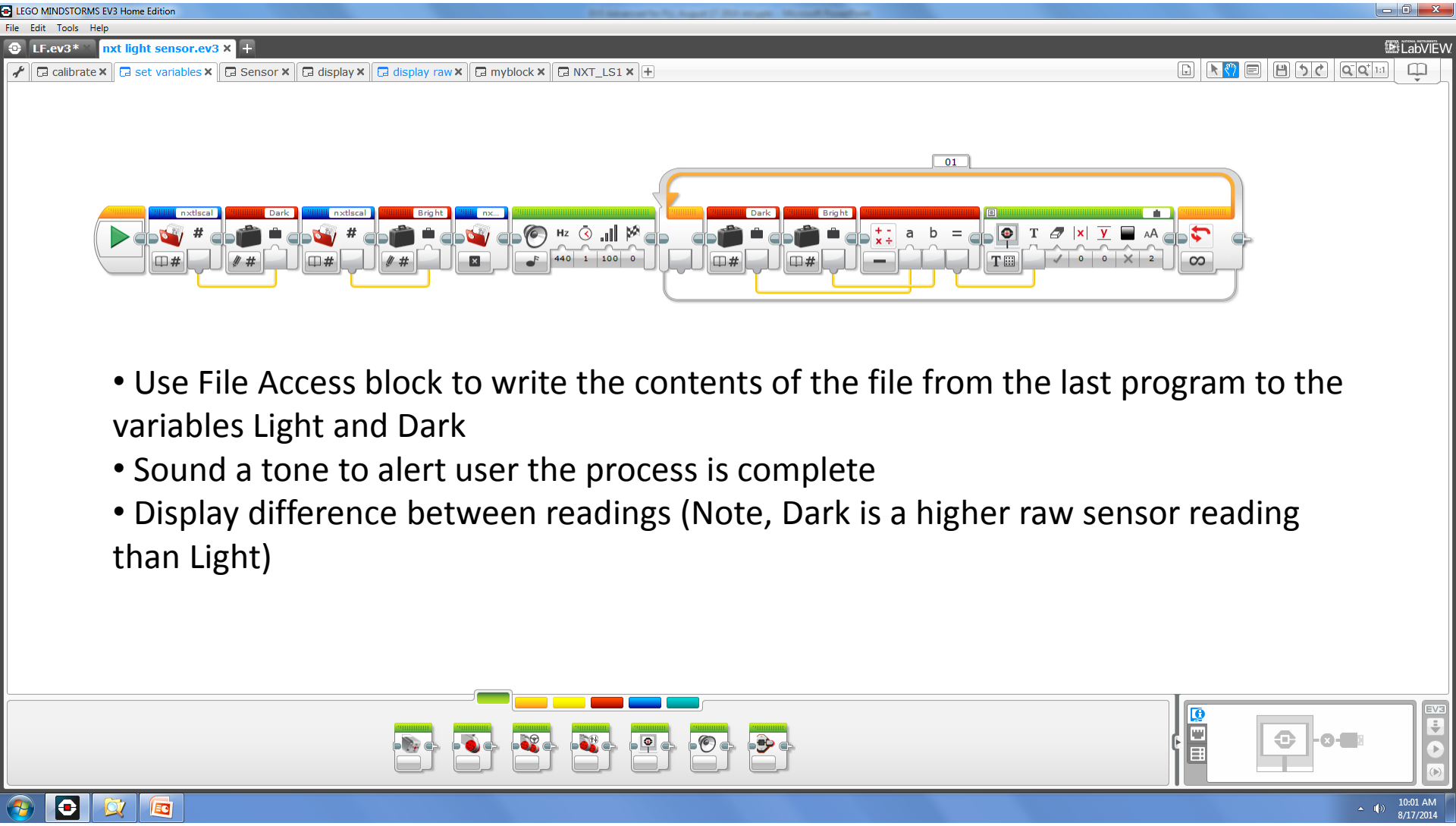
- File Access blocks
- Variable
- Raw Sensor Readings
- My Blocks

# Calibrating the NXT Light Sensor



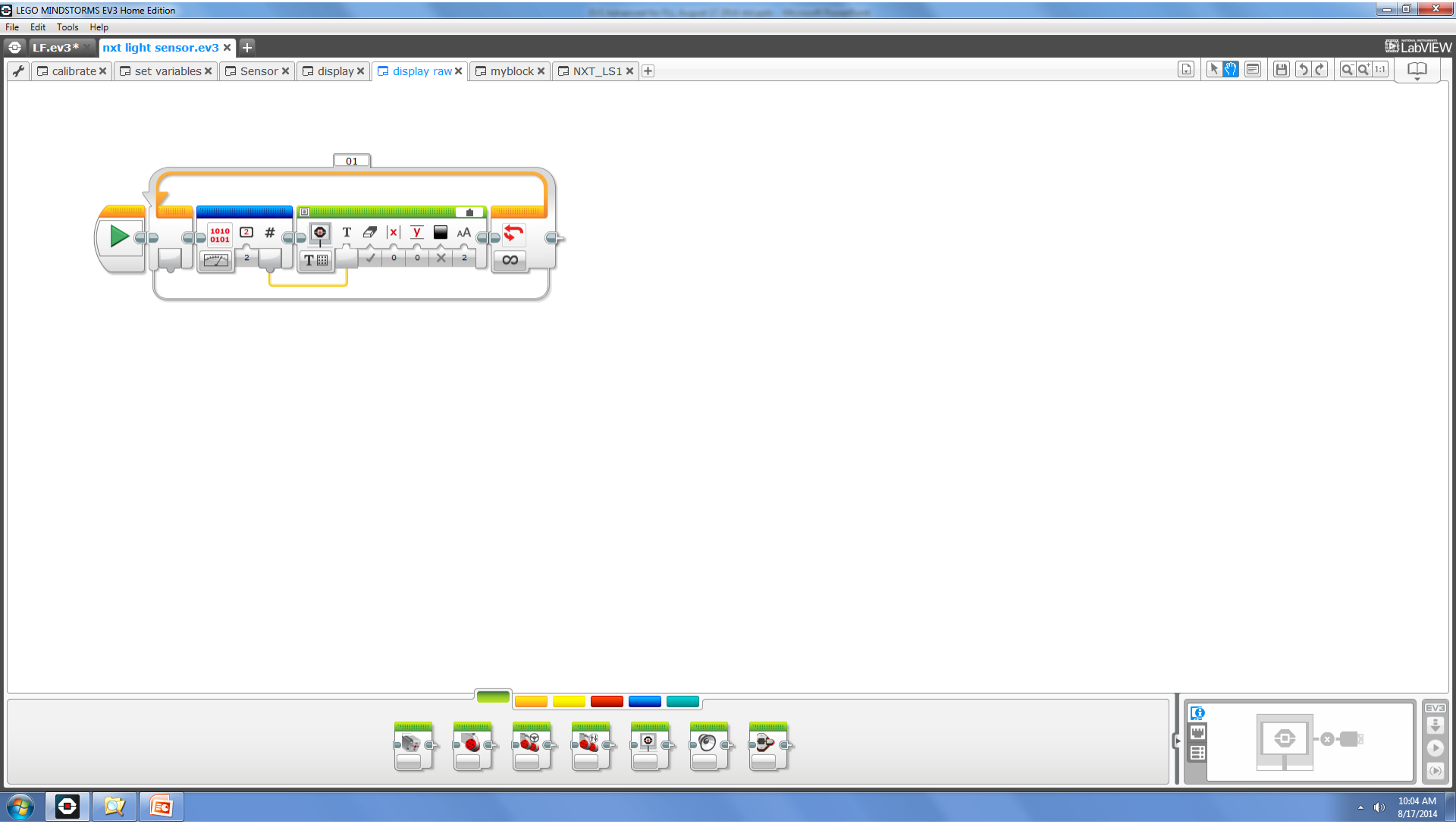

- File access block
- Read raw sensor values block

# Set Up Variables

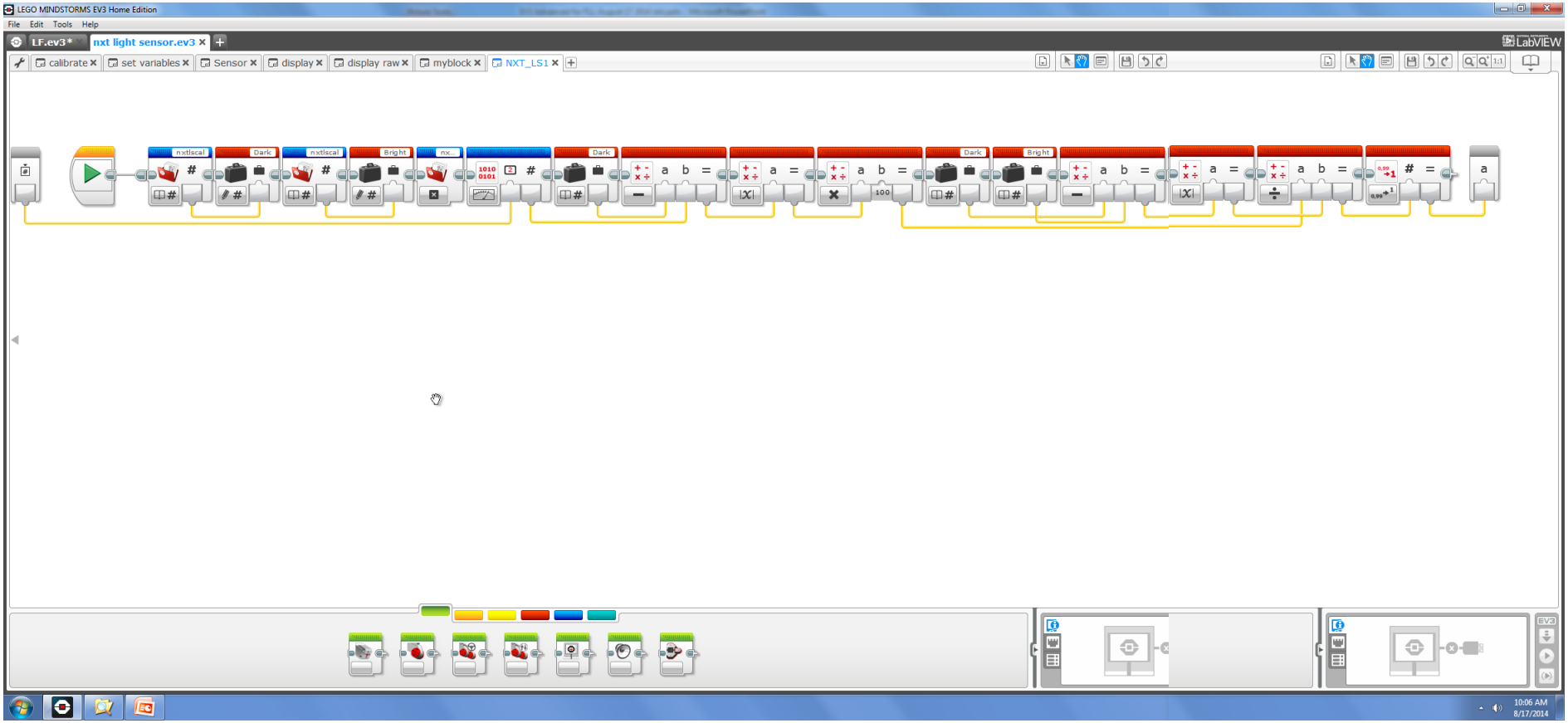



- Use File Access block to write the contents of the file from the last program to the variables Light and Dark
- Sound a tone to alert user the process is complete
- Display difference between readings (Note, Dark is a higher raw sensor reading than Light)

# Display a Raw Sensor Value



# The NXT Light Sensor My Block for EV3



# Display Your My Block!