ADVANCED EV3 PROGRAMMING LESSON

Squaring or Aligning on a Line

By Sanjay and Arvind Seshan

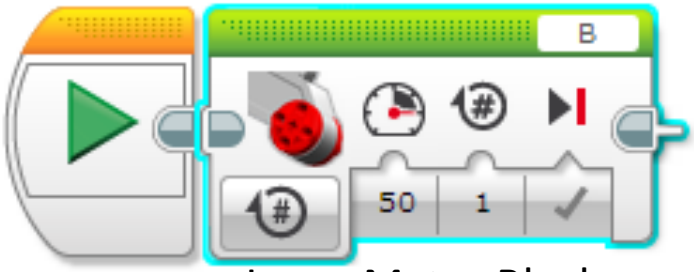
Lesson Objectives

- Learn how to get your robot to square up (straighten out) when it comes to a line
- Learn how squaring (also known as aligning on a line) can help the robot navigate
- Learn how to improve initial code for aligning by repeating a technique
- Practice creating a useful My Block

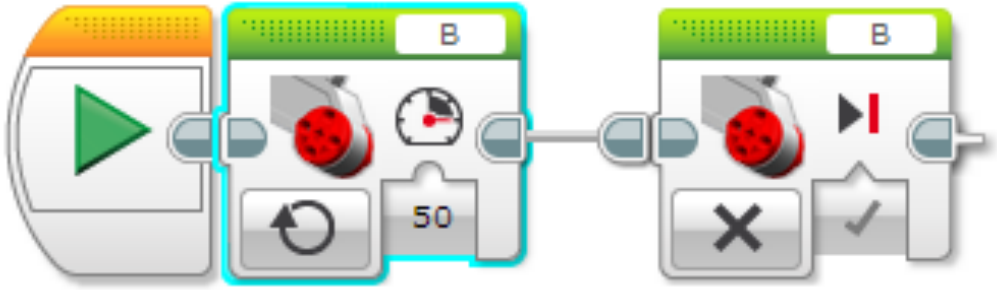
- Prerequisites: My Blocks with Inputs & Outputs, Data Wires, Parallel Beams, Parallel Beams Synchronization

Review: Motor Movements

- Move Steering lets you control both motors at the same time
- What if you want to move or stop one motor at a time?
  - Use the Large Motor Block



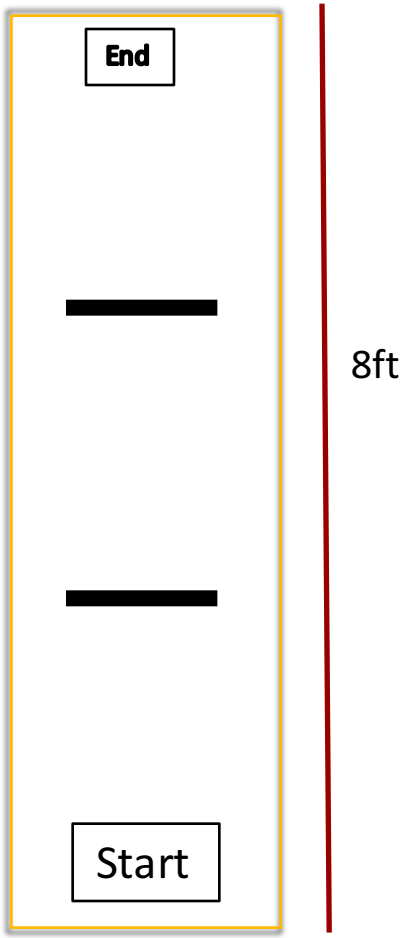
Large Motor Block



Large motor block in ON mode / OFF mode

Why Align on a Line?

- Aligning on a line helps the robot navigate
  - Robots get angled as they travel farther or turn (the error accumulates)
  - Aligning on a line can straighten out a robot.
  - Aligning can tell a robot where it is when it has to travel far
- Example Goal: Your robot must deliver an object only inside a small END area. The distance between start and end is 8 feet
  - Do you think your robot can travel 8 feet and continue to be straight?



Three Easy Steps to Align

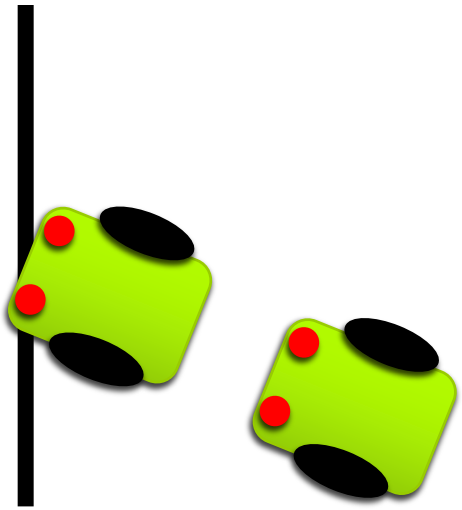
Challenge: Make the robot straighten out (align/square up)

STEP 1: Start both motors

STEP 2: Stop one motor when the sensor on the corresponding side sees the line

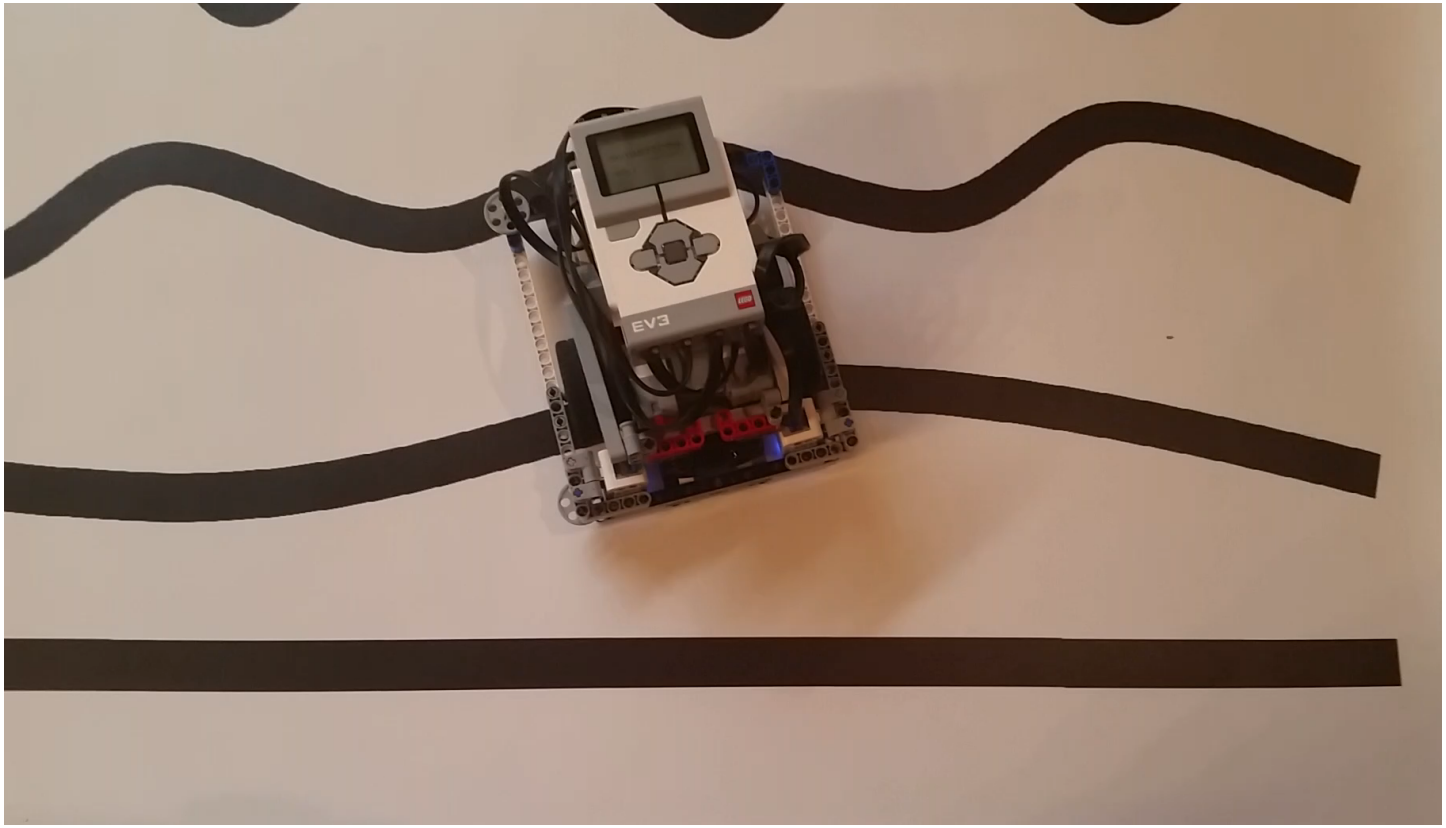
STEP 3: Stop moving the second motor when the sensor on that side sees the line

Hints: Use a Large Motor Block, Use Parallel Beams, Use the Large Motor Block



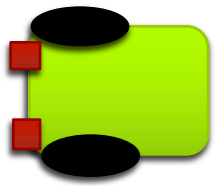
(This slide is animated)

What Aligning Should Look Like



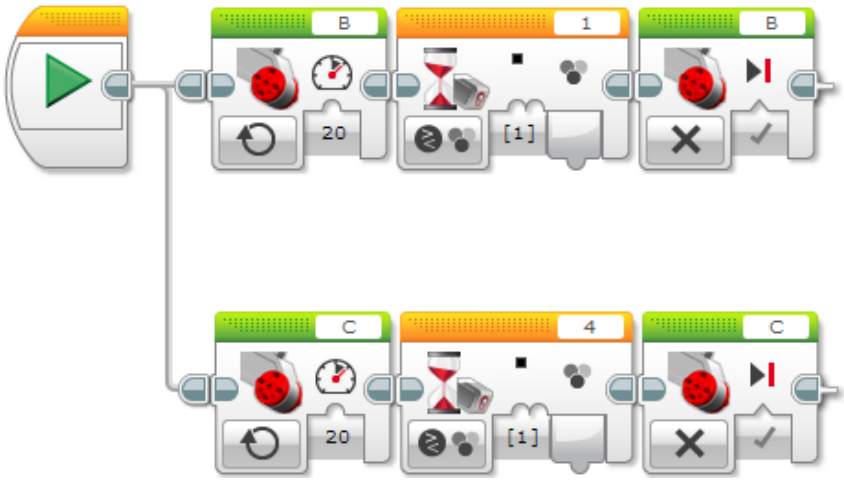
Notes About Our Solution:

- Our solution uses 2 Color Sensors (connected in Ports 1 and 4).
- Our solution assumes that the color sensor on port 1 is next to the wheel on motor port B and color sensor on port 4 is next to the wheel on motor port C.
- You should adjust the ports as needed
- Your color sensors should NOT be placed right next to each other (See red boxes below in robot image. These are the color sensors.)



Basic Solution: Moving Until Line

We use a parallel beam here so that we can do 2 simultaneous actions.
In the top beam, Motor B on our robot keeps moving until the Color Sensor on Port 1 sees Black. Then it stops.
In the bottom beam, Motor C on our robot keeps moving until the Color Sensor on Port 4 sees Black. Then it stops.



Step 1 Goal: Create a SIMPLE way to square up on the line

Note 1: You will need 2 EV3 Color Sensors (connected in Ports 1 and 4 in this program)

Note 2: This program squares onto a Black Line (you can change this to whatever color the EV3 accepts.

Note 3: This program uses the color sensor in COLOR MODE. You can write a program that uses LIGHT MODE, but you will have to calibrate your sensors. We will show you that in another lesson.

Note 4: Your robot design will make a difference - whether you have your color sensors in the rear or front of your robot, and how far apart the sensors are (the further apart, the better).

Note 5: You should adjust the ports as needed - e.g. this assumes that a color sensor on port 1 is next to the wheel on motor port B and color sensor on port 4 is next to the wheel on motor port C.

Note 6: While the robot will be on the black line, this will not create a perfect alignment. See instruction in Step 3 for a simple fix.

Note: Synchronization & Parallel Beams

- When you have two or more beams you do not know when each beam will finish.
- If you wanted to move after the align finishes you might try to add a move block at the end of one of the beams.
  - Note: This will not work because EV3 code will play your move block without waiting for the other beam to finish.
  - Solution: You need to synchronize your beams. To learn more about synchronization and solutions go to the Advanced EV3Lessons.com Lesson on Sync Beams
- The problem of synchronization can also be solved by making a My Block out of the align code (refer to My Block lesson in Intermediate)
  - My Blocks always wait for both beams to finish before exiting

Improving Your Align Code

- What do you notice about the solution we just presented?
  - The robot isn't quite straight (aligned) at the end of it.
  - Both color sensors are on the line, but the robot stops at an angle.
- Challenge Continued: Think about how you can improve this code so that the robot ends straighter

Tips for Success

- You will get better results
  - ….if your color sensors are about 4mm-12mm from the ground (see Color Sensor Placement Lesson in Robot Design Lessons)
  - ….if you don't come at the line at steep angles
  - ….if you keep your color sensors spread apart

Credits

- This tutorial was created by Sanjay Seshan and Arvind Seshan
- More lessons at www.ev3lessons.com

ADVANCED EV3 PROGRAMMING LESSON

Arrays

By Sanjay and Arvind Seshan

Lesson Objectives

- Build upon skills from the Variables lesson in Intermediate
- Learn how to read/write to arrays
- Learn about the Array Operations block
- Learn to use the loop count in a loop

- Prerequisites: Data Wires, Loops, Variables

Why Use Arrays?

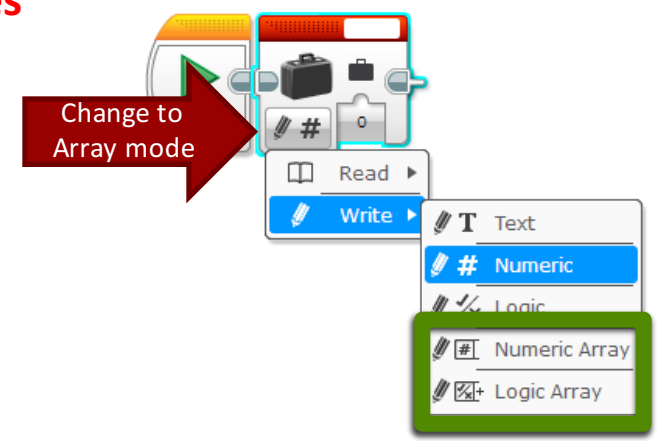
1. Simplify programs by storing multiple related values in a single variable
2. Can be used with loops to make compact and useful programs
3. Are useful for making a custom calibration program (see NXT Light Sensor in EV3 on our contributed lessons tab)

Arrays

- What is an array?
  - An array is a variable that holds multiple values
- There are two types of arrays:
  - Numeric Array (Holds a set of numbers … 1,2,3,10,55)
  - Logic Array (Holds a set of logic … True, True, False)
- They can be used as either Inputs or Outputs so you can either….
  - Write – put a value(s) into the array
  - Read – get the value(s) from the array out

Array Blocks: Quick Guide

Modes



Key

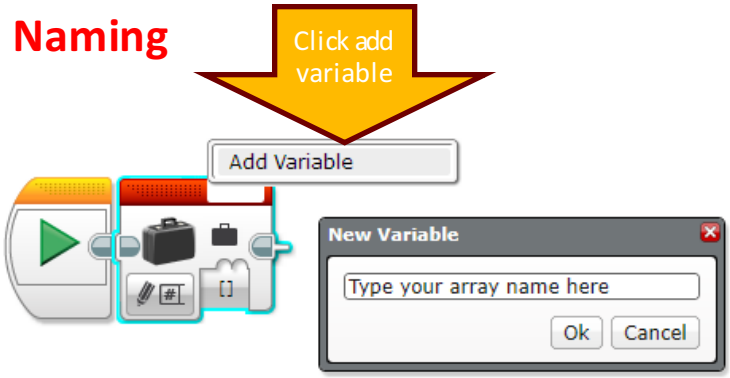
Logic Array

Numeric Array

Write (Inputs) have 2 bumps up

Read (Outputs) have 2 bumps down

Naming



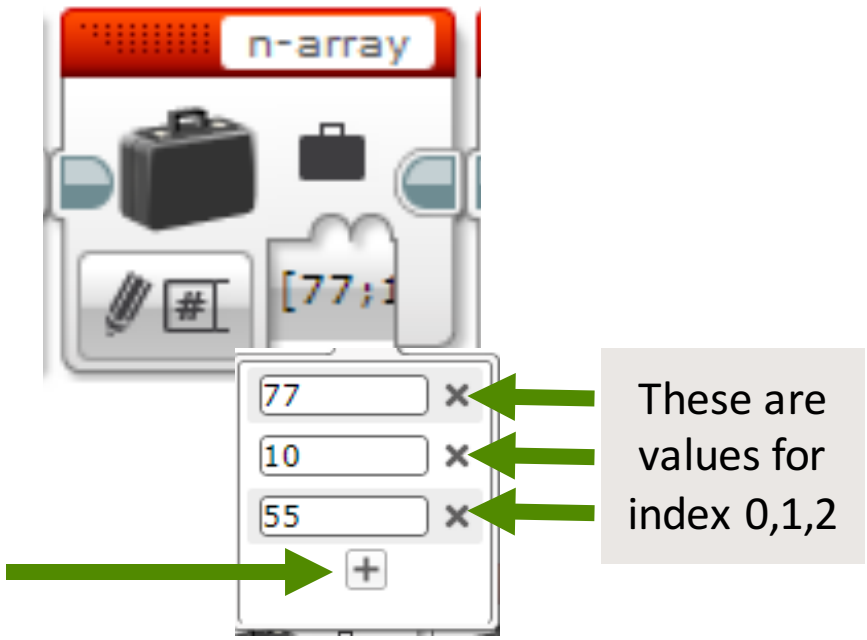
Quiz

Read logic array	Write logic array	Read numeric array	Write numeric array

Identify if the variables are Inputs/Outputs and if they are Numeric/ Logic

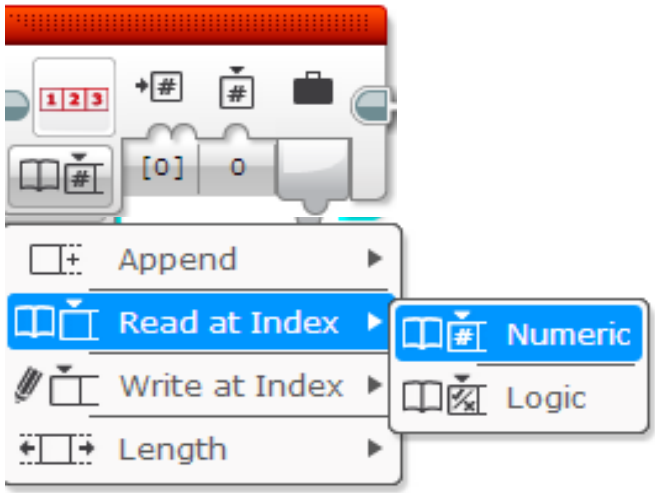
Array Indexes

- Each value in an array is assigned an index
- The first value would be at index 0
- Logic arrays would store True/False instead of numbers
- To add a value to an array click the plus +
  - This adds an entry at the next index value (i.e. index 3)

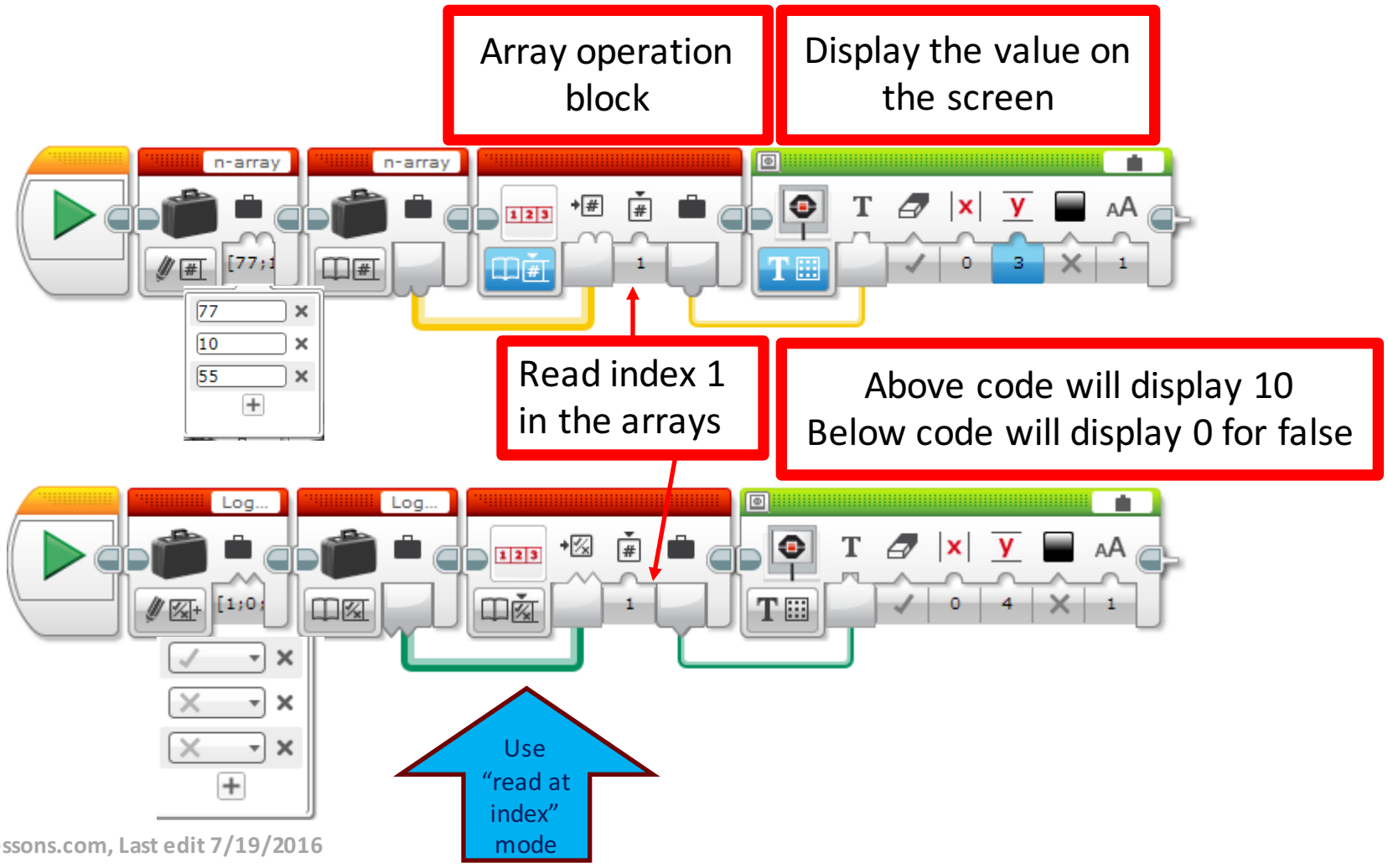


Block: Array Operations

- This block is used to read or write to Logic or Numeric arrays
- Different modes:
  - Append: Add a new entry after the last array index
  - Read at index: Reads the value at a certain index
  - Write at Index: Write a new value to a certain array index
  - Length: How many entries are in the array
- Both write and append output an array → you will need to write this array back to the variable if you wish to update the stored array (see write/append slides)



How do you use Arrays (Reading)?



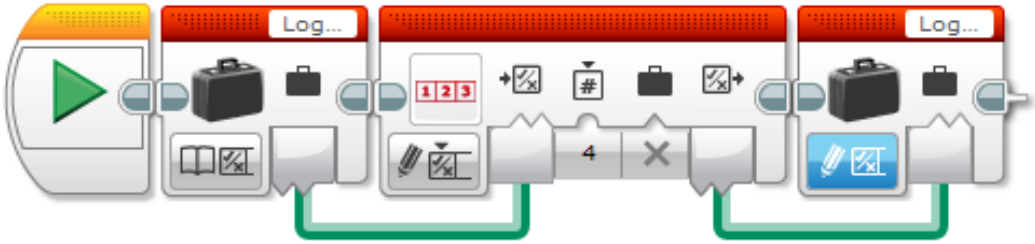
How do you use Arrays (Writing)?

This will write 700 to array at index 4

Read the array you want to write to

Use array operations to write a value to a certain index

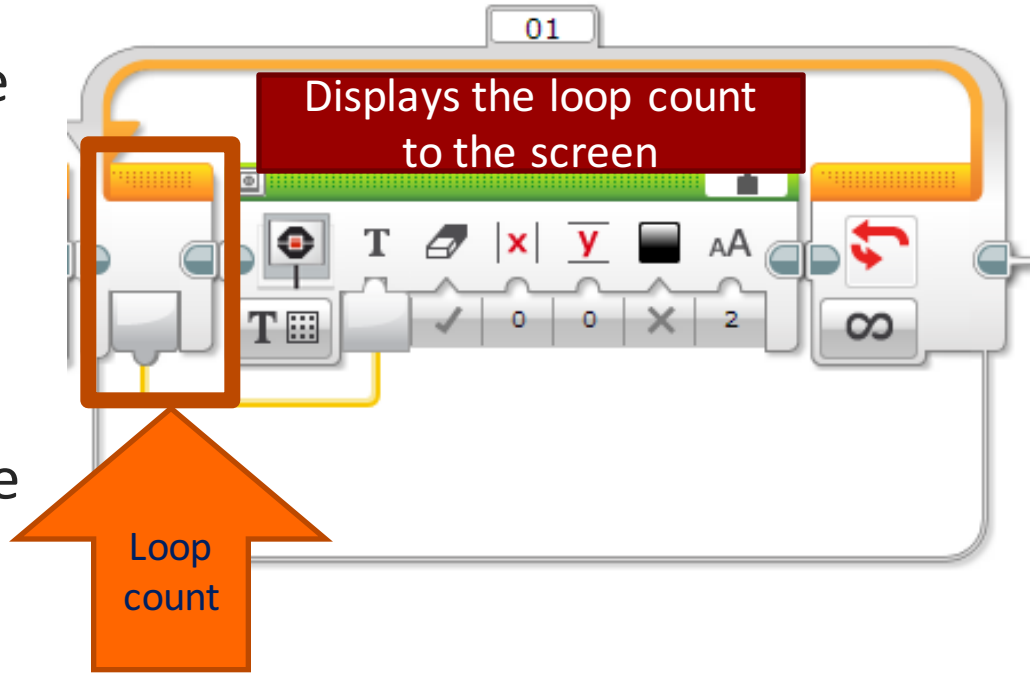
Write the output back to the array



This will write False to array at index 4

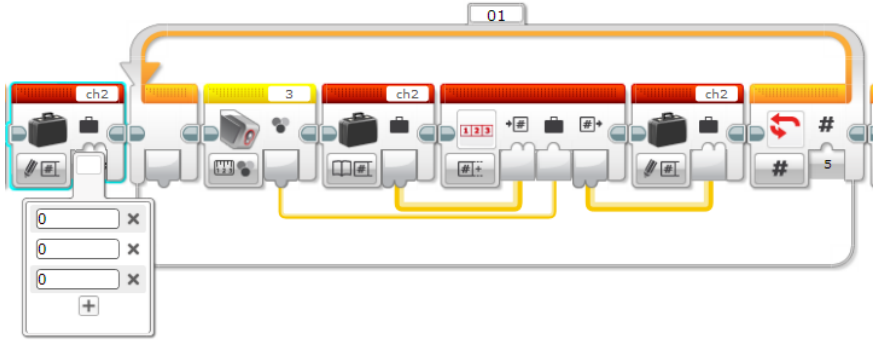
Block Setting: Loop Count

- The loop count outputs the amount of times the blocks inside the loop have played.
- This is useful to create a program that runs different code every time it goes in the loop
- It is also useful for computing on each item of an array



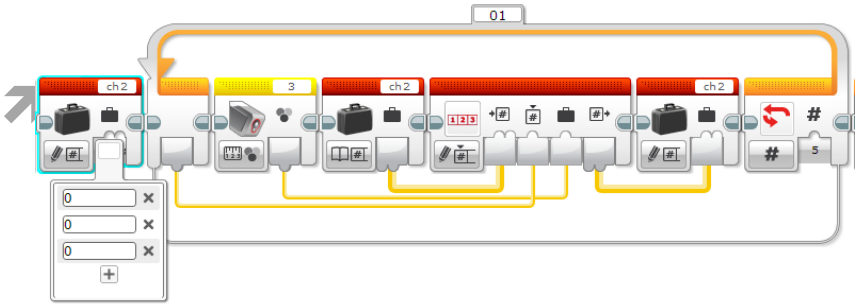
Note: Append vs. Write

- Append adds entries to the end of an array (i.e. creates a new index value)



- This code produces an array with 8 entries (three 0's followed by 5 light readings)

- Write overwrites the entry at the chosen index

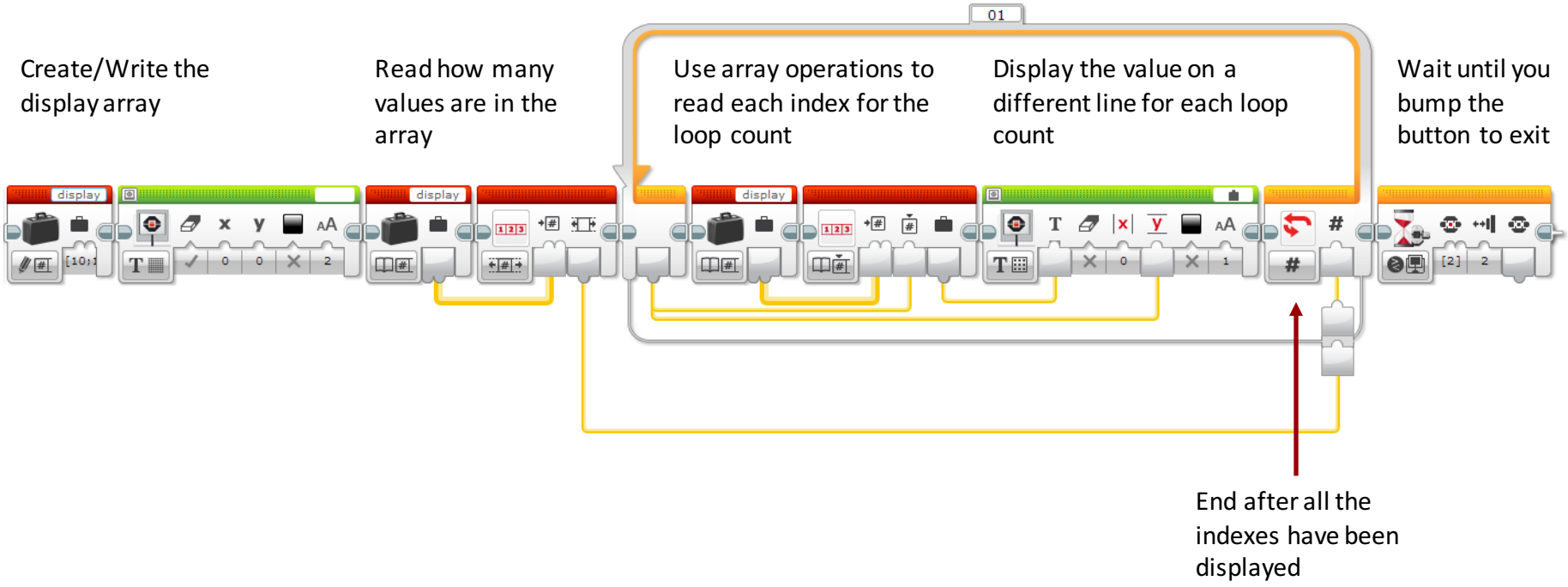


- This code produces an array with 5 entries (just 5 light readings)

Challenge 1

- Make a program that displays all the entries of an array. Display each index on a different line. You can use only one display block.
- Tips: You will need to use loops, loop count, array block, array operations

Challenge 1 Solution



Challenge 2

- Make a program that adds up all the entries of an array. Display the sum.
- Tips: You will need to use loops, loop count, array block, array operations

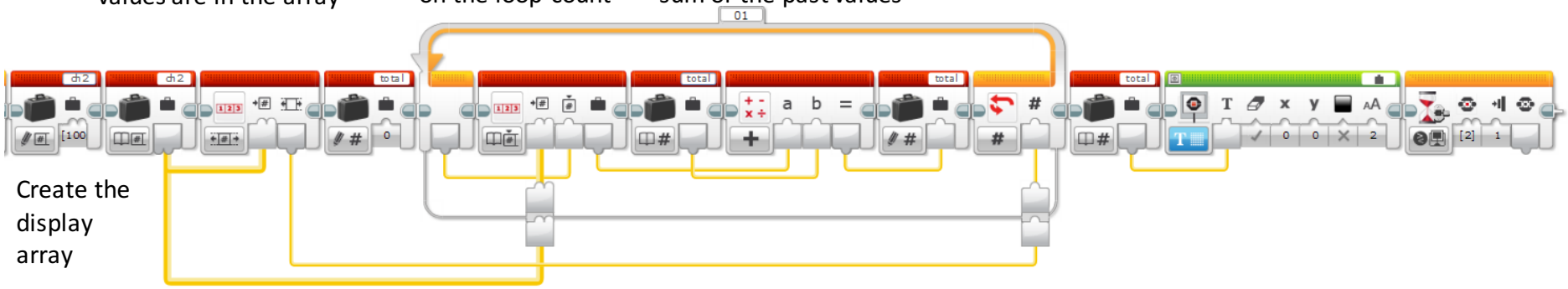
Challenge 2 Solution

Read how many values are in the array

Read the index based on the loop count

Add the array value to the sum of the past values

Display to the screen



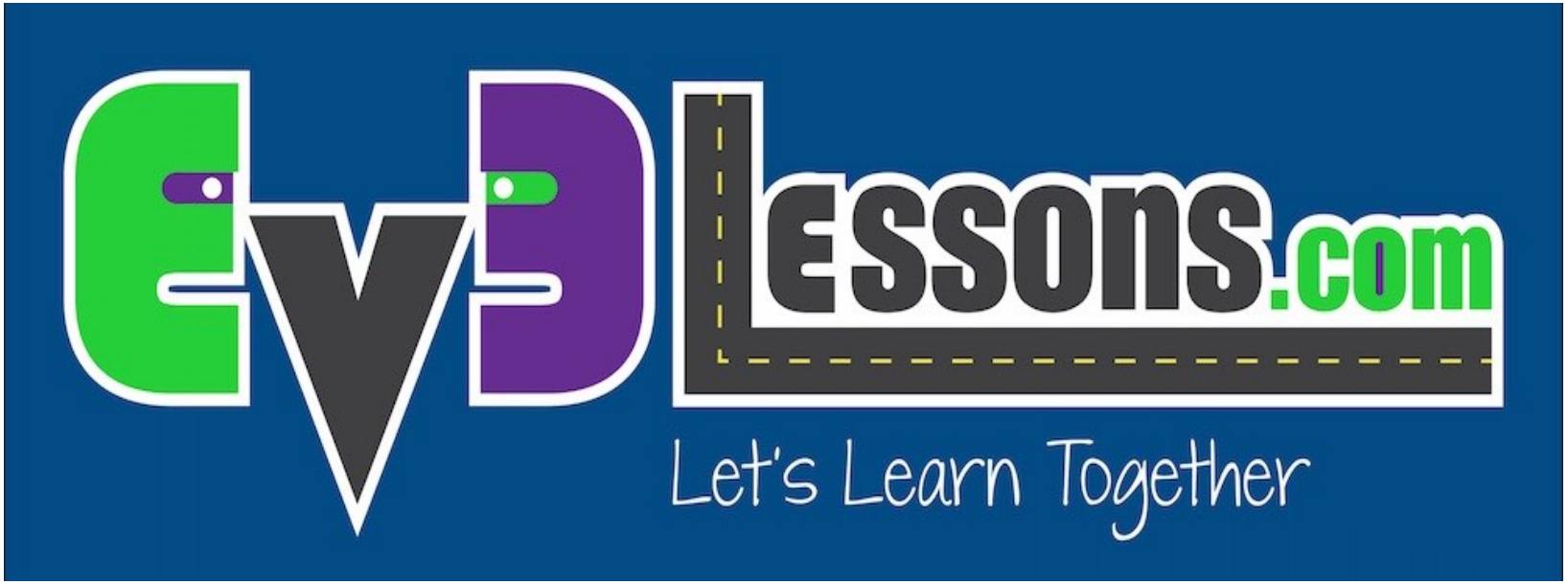
Create the display array

Next Steps

- Here are some fun things to try:
  1. Make a program to compute the average value in an array
  2. Make a program that always saves the last 4 light sensor readings in an array
  3. Create an array that stores calibration values for each sensor port

Credits

- This tutorial was written by Sanjay Seshan and Arvind Seshan
- More lessons at www.ev3lessons.com



Basic Line Follower

By Sanjay and Arvind Seshan

BEGINNER PROGRAMMING LESSON

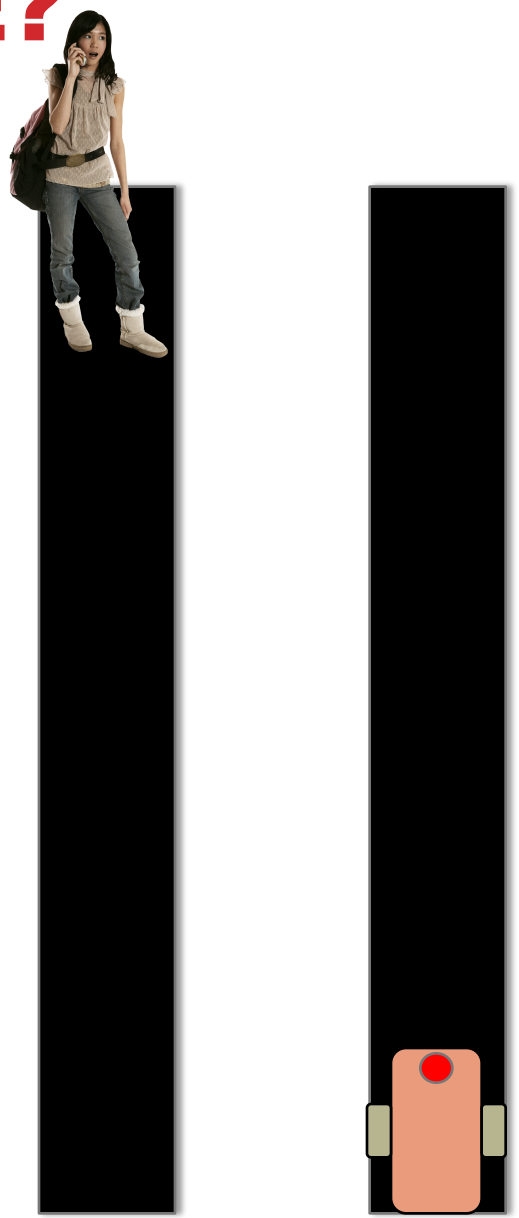
LESSON OBJECTIVES

1. Learn how humans and robots follow lines
2. Learn how to get a robot to follow a line using Color Mode on the EV3 Color Sensor
3. Learn how to follow a line until a sensor is activated
4. Learn how to follow a line for a particular distance
5. Learn how to combine sensors, loops and switches

TEACHER INSTRUCTIONS

- Slides 4-7 are animated. For students to better understand how a line follower works and how a human and a robot follow a line, we recommend that you play the animation
- Give each student/team a copy of the worksheet.
- Challenge 1 begins on slide 10 and Challenge 2 on Slide 13
- Discussion Guide is on Slide 16
- More advanced students might be interested in other line followers on EV3Lessons.com

FOLLOW THE MIDDLE?

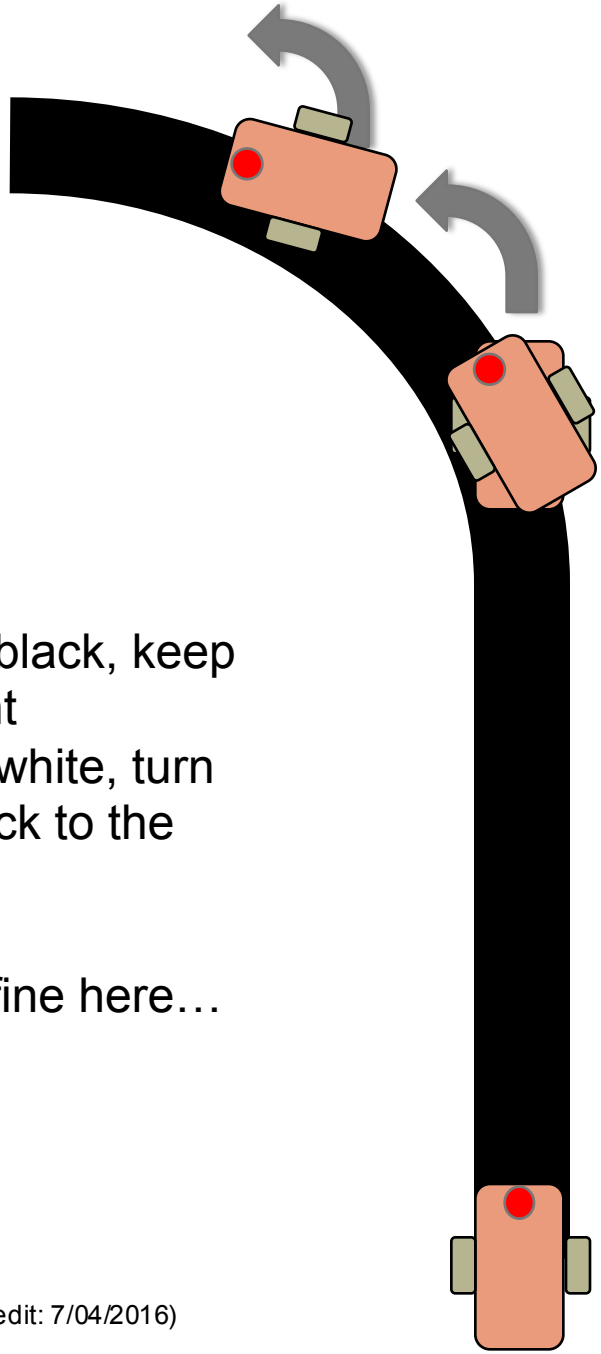


Humans want to follow the line in the middle.

Let's have the robot do the same thing using the Color Sensor

What type of questions can we ask using this sensor

- Are you on line or not?



1. If we are on black, keep going straight
2. If we are on white, turn left to get back to the line

Seems to work fine here…

1. If we are on black, keep going straight
2. If we are on white, turn left to get back to the line

OH NO… my robot is running away….

When the robot leaves the left side of the line, the program no longer works!

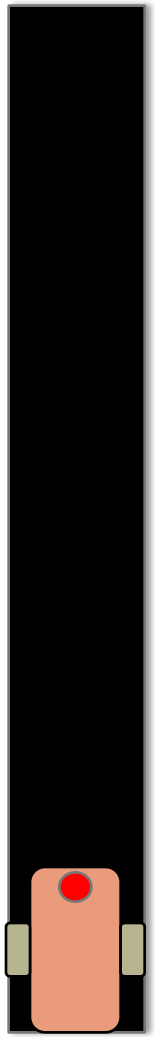
LINE FOLLOWING: ROBOT STYLE

Why could the Human follow the middle?:

- They can see ahead.
- They can see the whole line and its surroundings
- They see both sides and which side they left

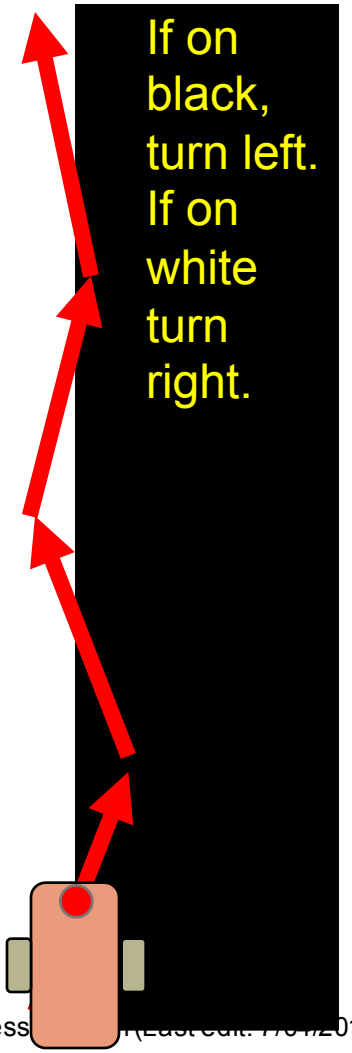
Why can't the Robot do the same thing?:

- Can't tell right or left side of the line
- How do we make sure the robot always veers off on the SAME SIDE of the line?
    - Instead of the middle, could the robot follow the "edge"?
- So now the robot will fall off only the same side.
- We will now show you how this works!



ROBOT LINE FOLLOWING HAPPENS ON THE EDGES

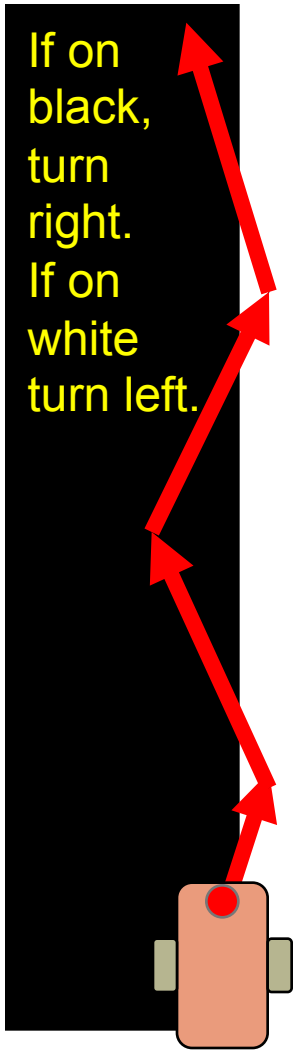
Left side line following



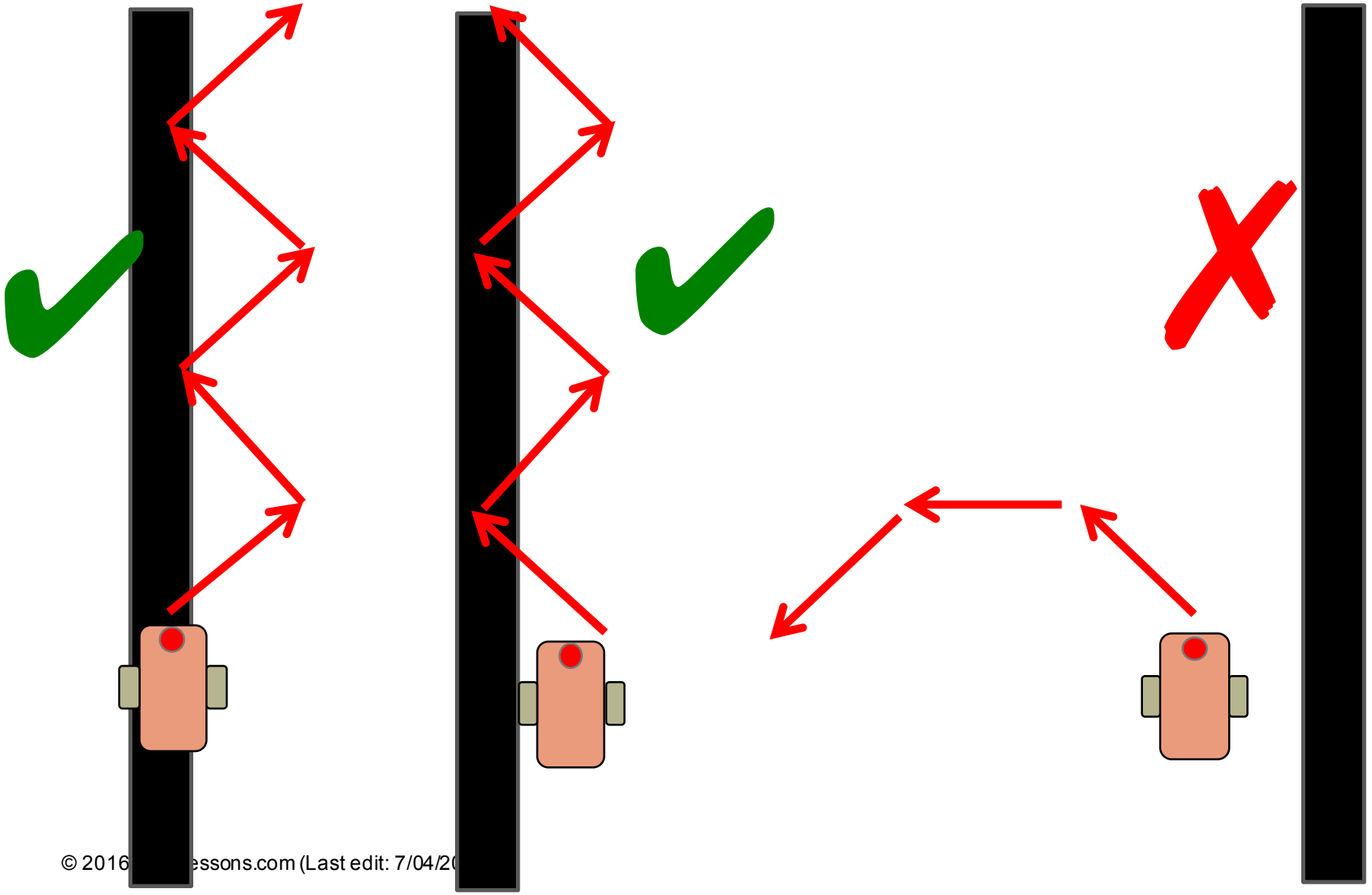
The robot has to choose which way to turn when the color sensor sees a different color.

The answer depends on what side of the line you are following!

Right side line following



STARTING THE ROBOT ON THE CORRECT SIDE



LINE FOLLOWER CHALLENGE 1

Step 1: Write a program that follows the RIGHT edge of a line.

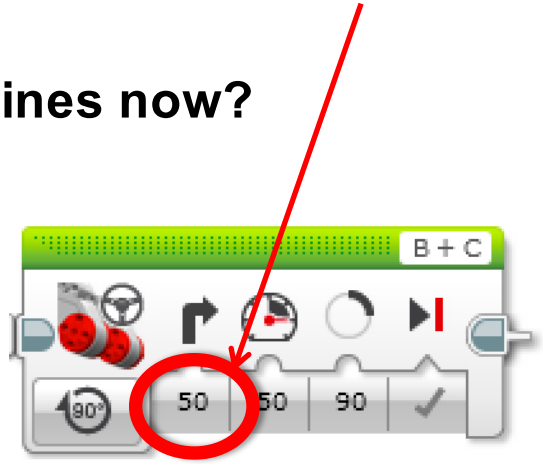
Hints: If your sensor sees black, turn right. If your sensor sees white, turn left. Use loops and switches!

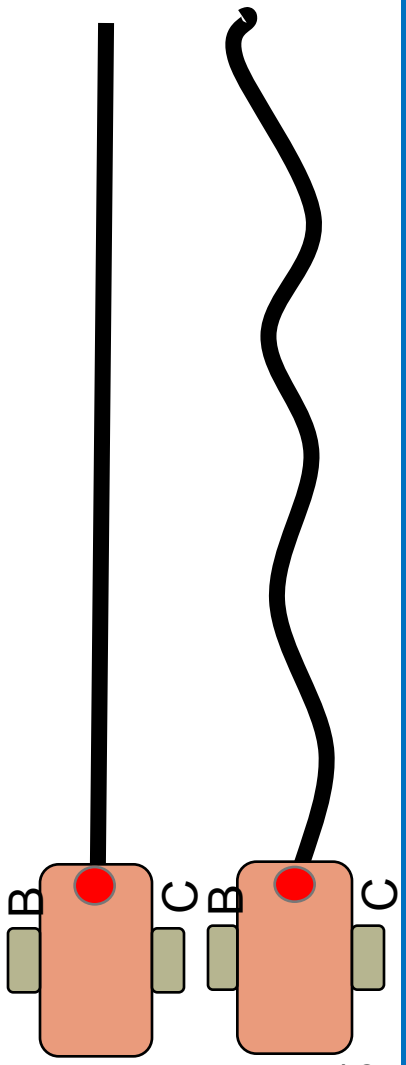
Step 2: Try it out on different lines.

Did your line follower work the same on straight and curved lines?

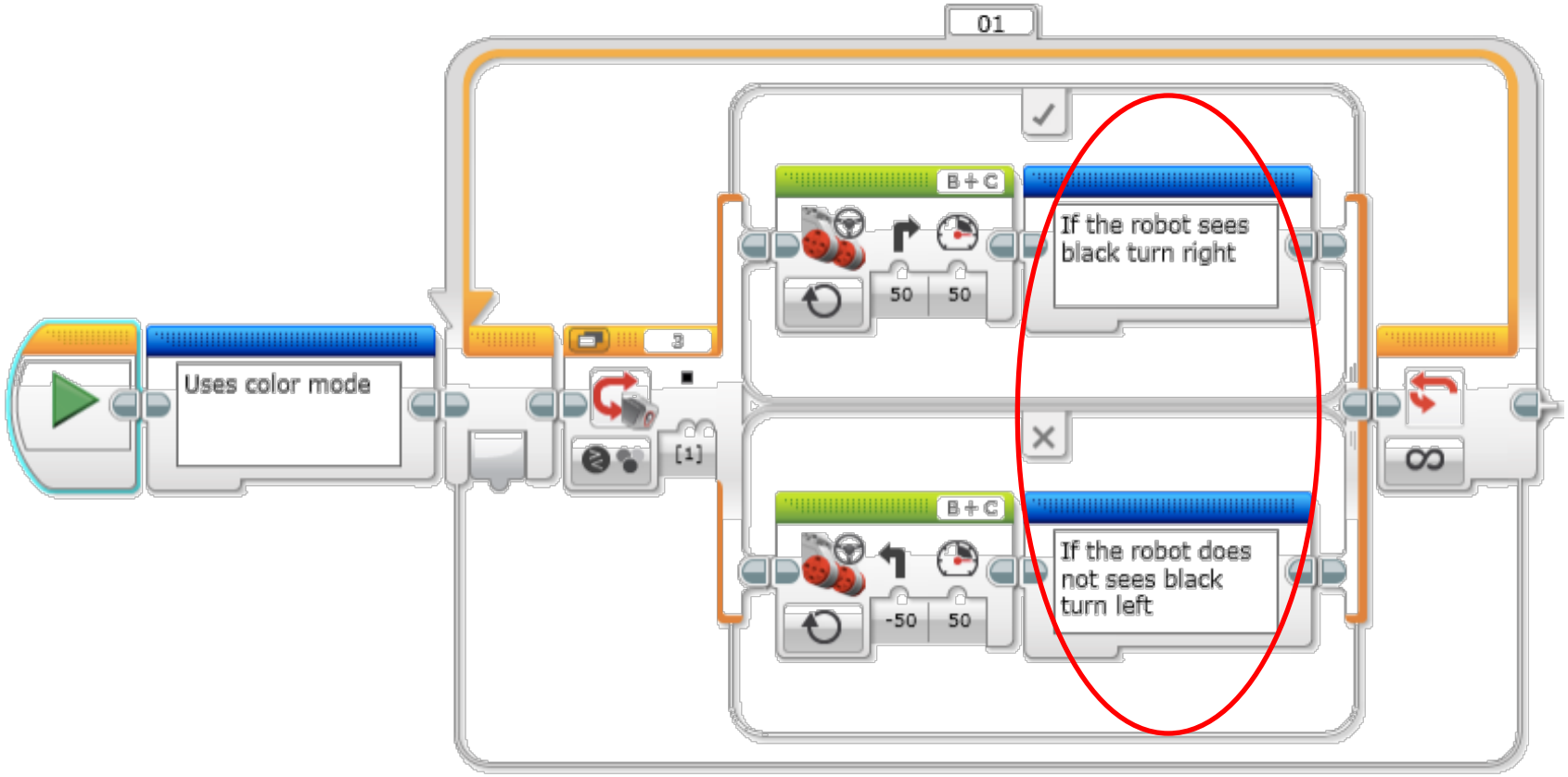
Step 3: If not, instead of turn Steering = 50, try smaller values.

Is it better on the curved lines now?



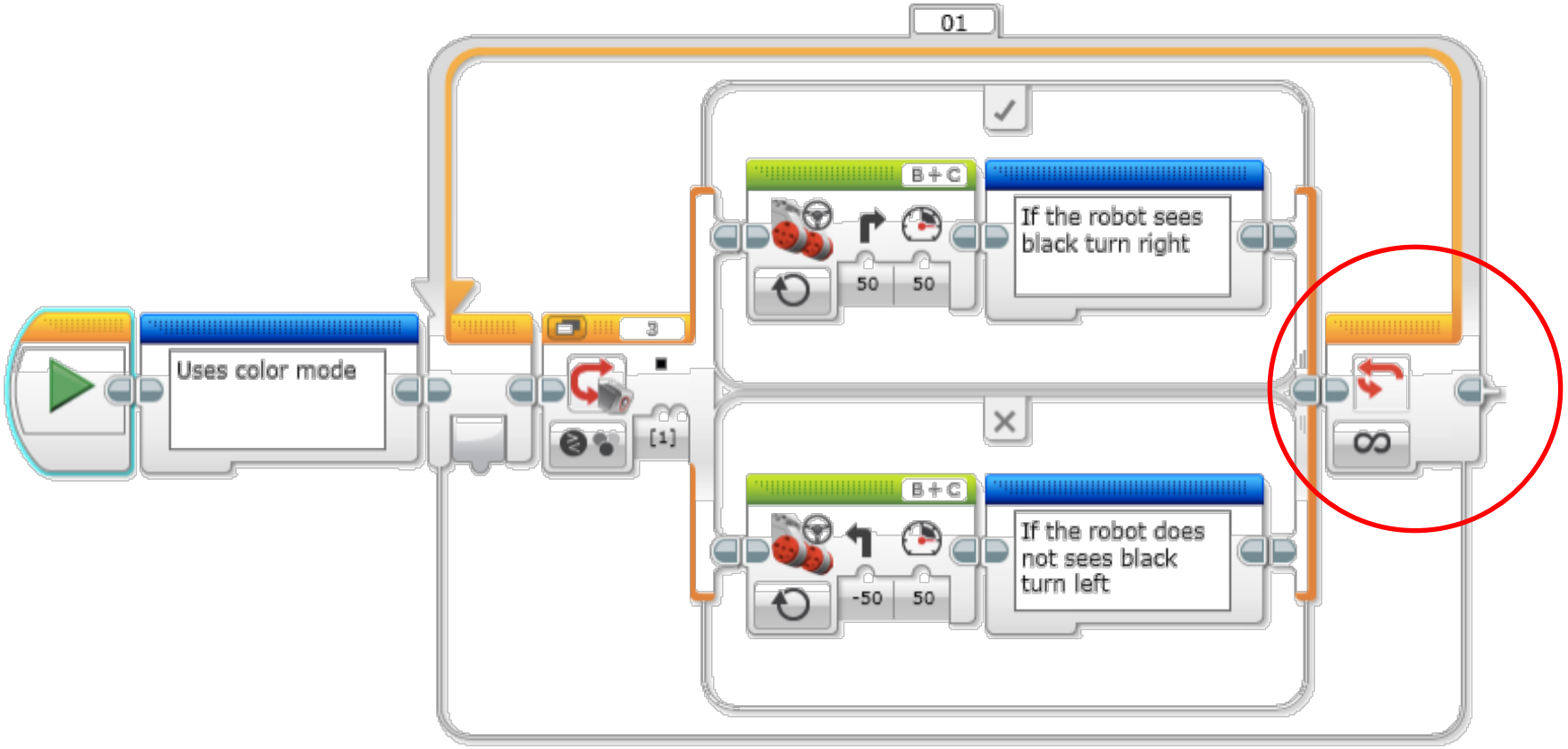


LINE FOLLOWING CHALLENGE SOLUTION



Q. Does this program follow the Right or Left side of a line?
A. The robot is following the Right Side of the line.

CHALLENGE 1 SOLUTION



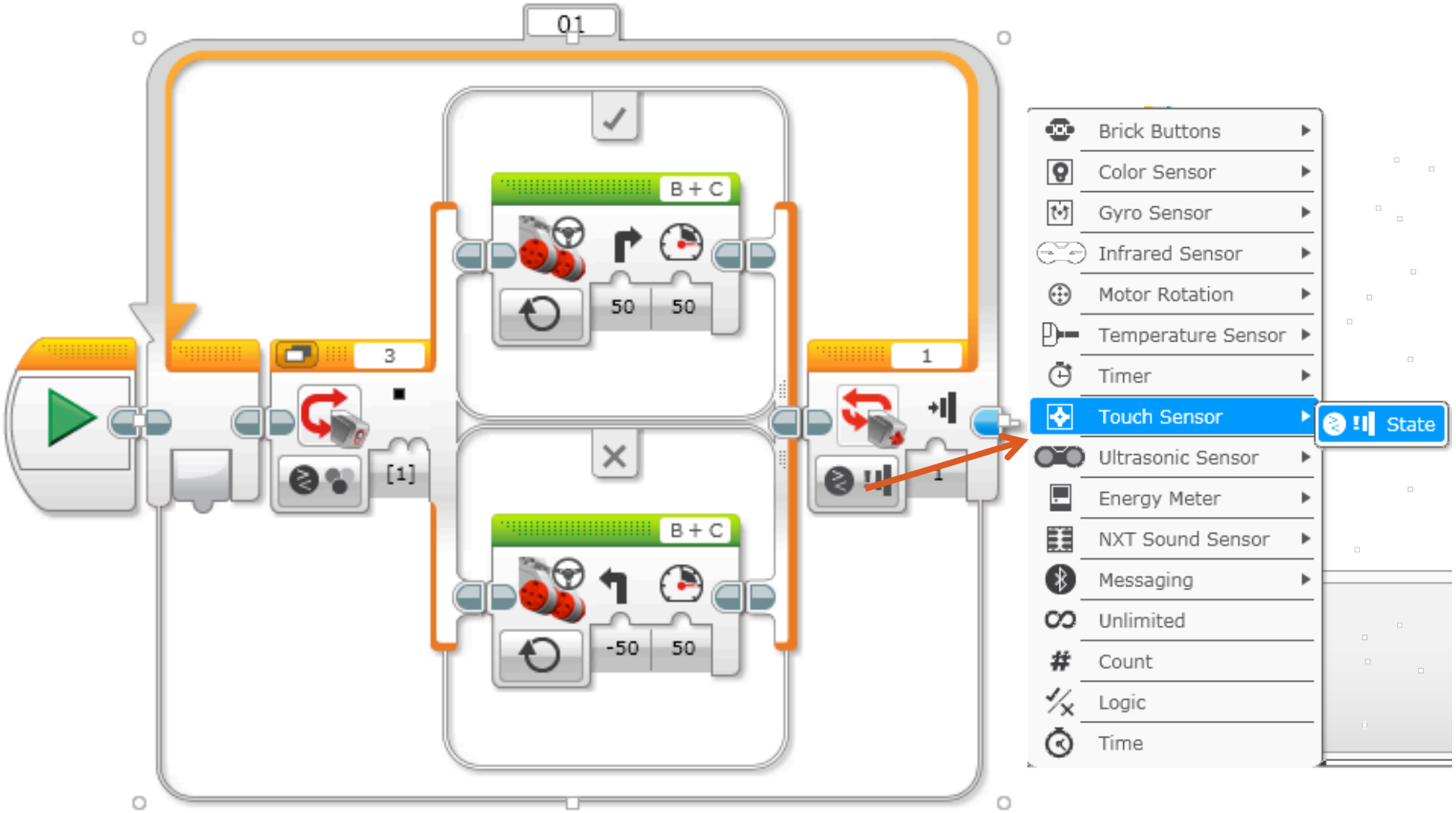
Q. This line follower goes forever. How do we make this stop?
A. Change the end condition on the loop.

LINE FOLLOWER CHALLENGE 2

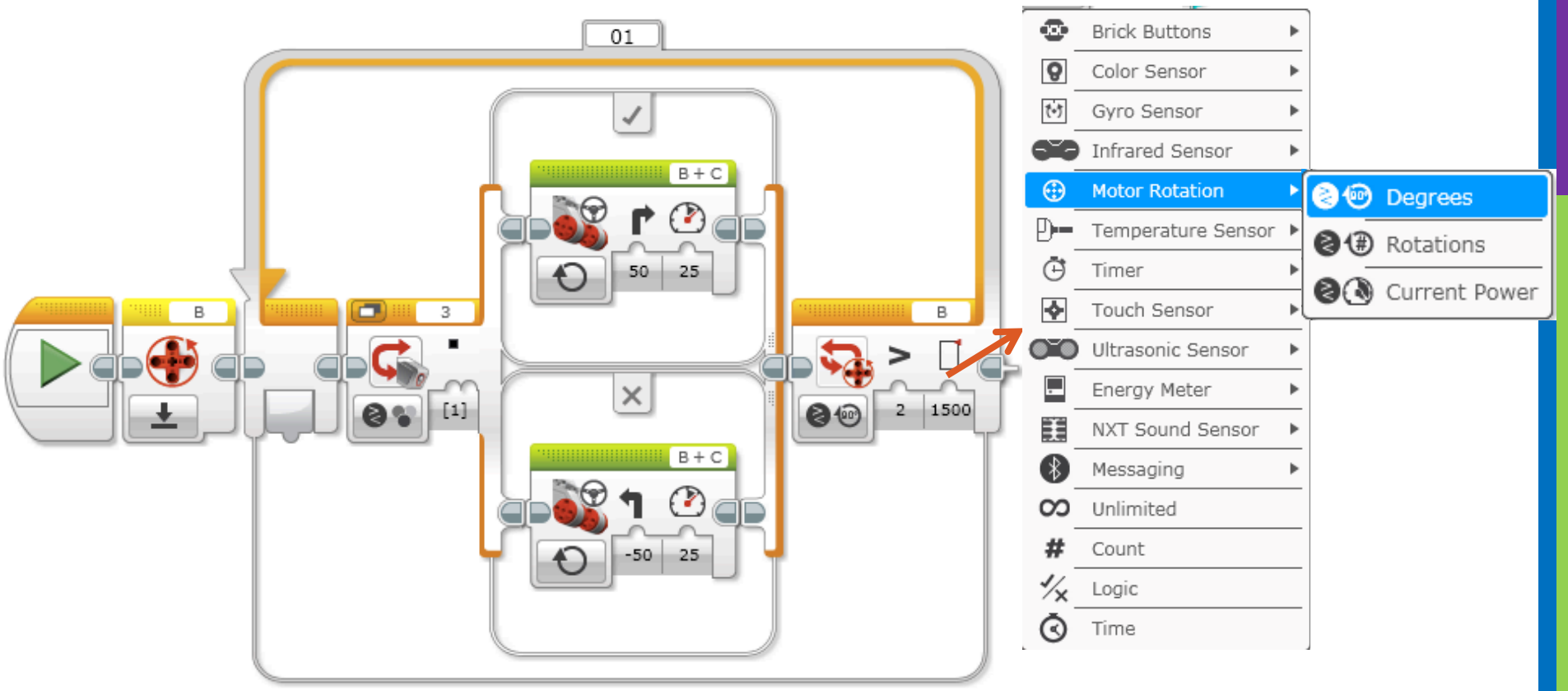
Part 1: Make a line follower that stops when you press the touch sensor

Part 2: Make a line follower that stops after it travels a particular distance

CHALLENGE 2 SOLUTION: SENSOR



CHALLENGE 2 SOLUTION: PARTICULAR DISTANCE



DISCUSSION GUIDE

Why is it important for the robot to follow the same side of the line?

The robot only knows to check if it is on or off the line.

This is a basic line follower. What are some things that were not good about this line follower? Do you think the line follower can be improved?

It wiggles a lot. Smoother line followers are described in the Advanced lessons

What sensor measures how far you have travelled?

The rotation sensor used in Challenge 2 solution measures how much the wheels have turned

How would you write a line follower that will stop when it sees a line? Or another color?

Change the loop exit condition to use the color sensor.

CREDITS

- **This tutorial was created by Sanjay Seshan and Arvind Seshan**
- **More lessons are available at www.ev3lessons.com**

BEGINNER FLL PROGRAMMING WORKSHOP

BY DROIDS ROBOTICS & EV3LESSONS

GOALS FOR THIS WORKSHOP

- **Use sensors to solve FLL missions**
  - Wait For Block with a Sensor
  - Line Following with Loops & Switches

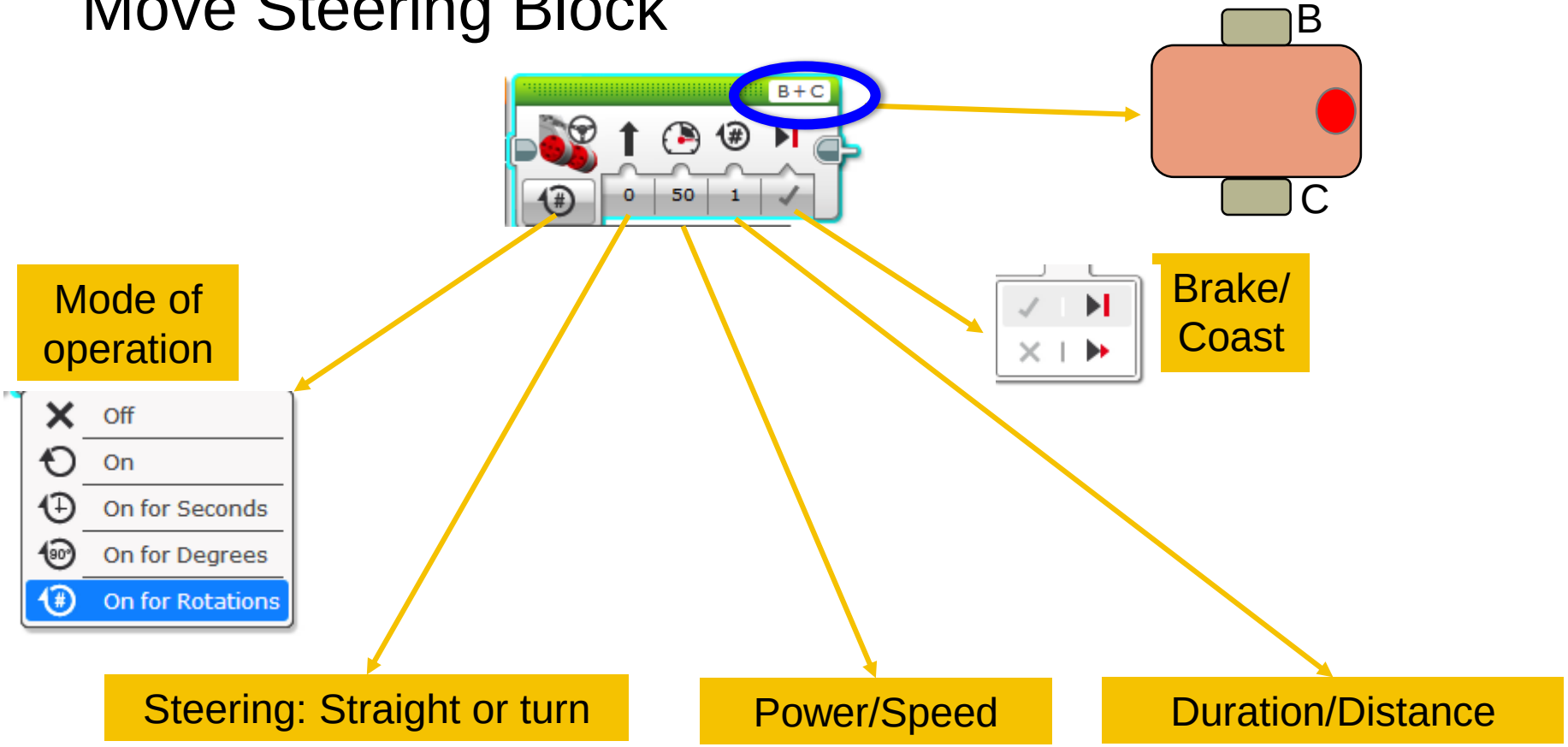
- **Learn some tips & tricks in building and programming**
- **Understand where on the Trash Trek Mat you can use these techniques**

WHERE CAN I LEARN MORE?

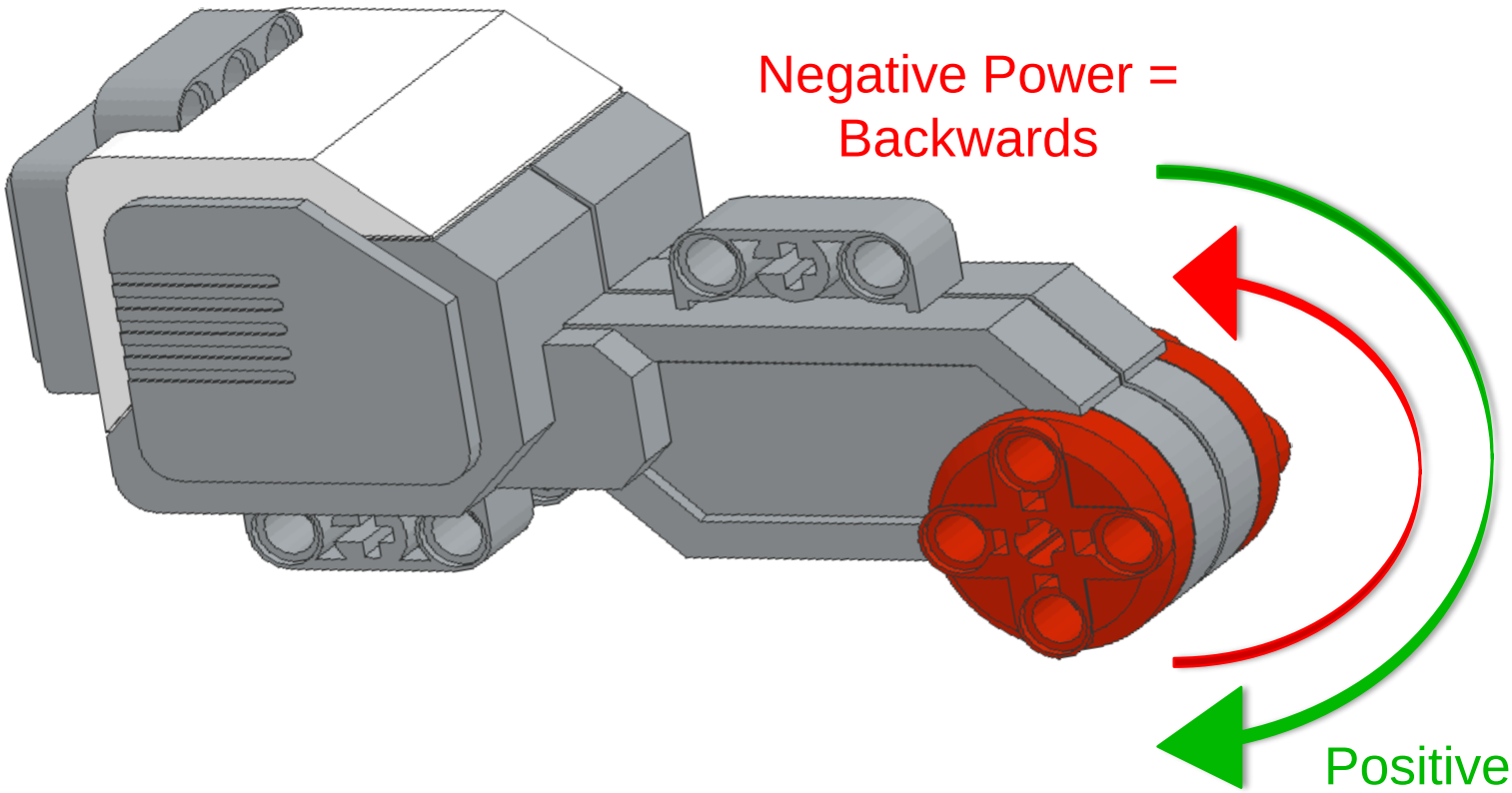
- **During the workshop, we only provided a review of Moving Straight, Turning and Port View. If you want to learn more, please visit EV3Lessons.com – Beginner.**
- **We only provided a quick introduction to Loops and Switches during the Line Following lesson. For fun challenges and to learn more, please view the Beginner lessons on Display Blocks, Loops and Switches.**

REVIEW: MOVING STRAIGHT

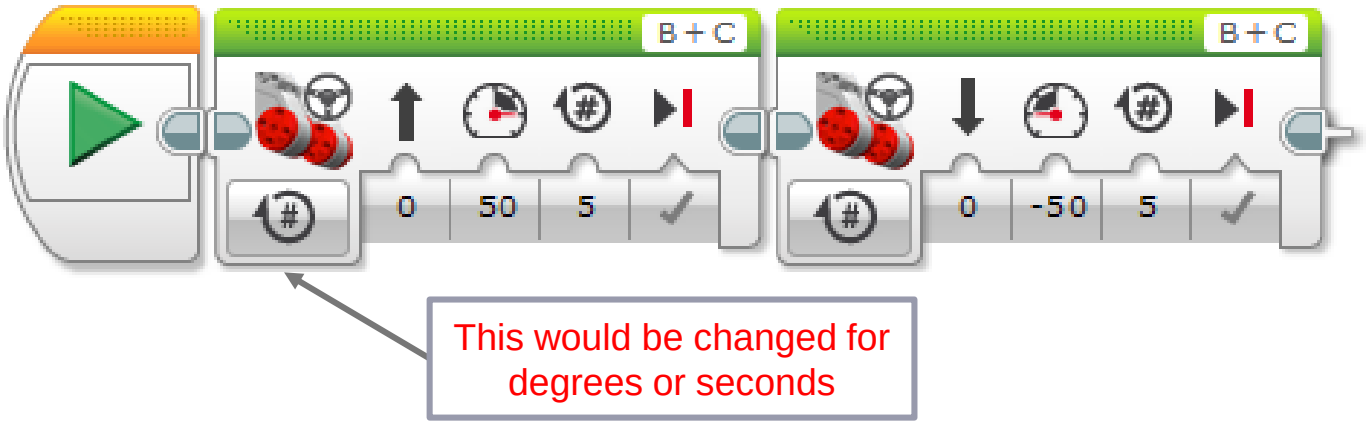
Move Steering Block



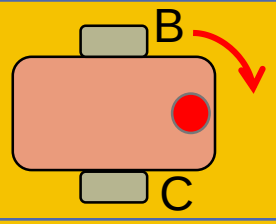
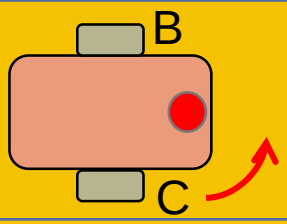
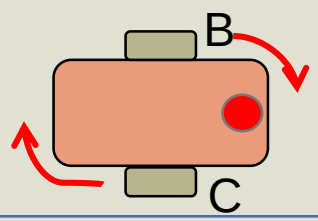
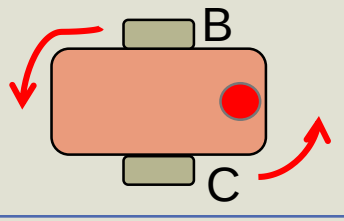
NEGATIVE & POSITIVE POWER: BACKWARD & FORWARD



REVIEW: MOVING FORWARD AND BACKWARDS

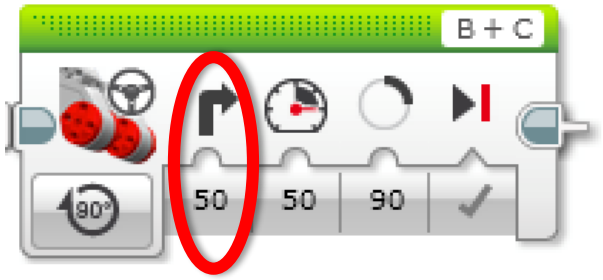


REVIEW: PIVOT & SPIN TURNS

Different Turns and Different Steering Values			
50	-50	100	-100
			
Pivot Turn Right	Pivot Turn Left	Spin Turn Right	Spin Turn Left
More accurate, but takes more space		Good for tight spaces, faster, less accurate	

Change Steering value

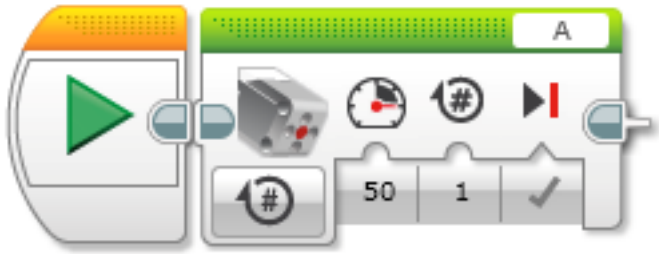
Move Steering Block



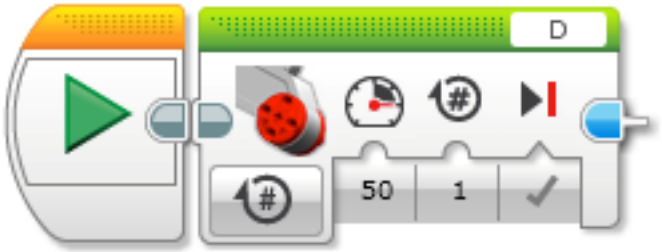
REVIEW: USING ATTACHMENTS

- **Attach a medium motor to Port A or a large motor to Port D as needed.**
- **Move Steering vs. Motor Block**
  - For moving your wheels you should use a Move Steering Block that syncs both wheel motors (see intermediate lesson called Move Blocks to learn about sync)
  - For moving your attachment your arm, you use either a Medium Motor Block or a Large Motor Block because you don't need to sync your motors.

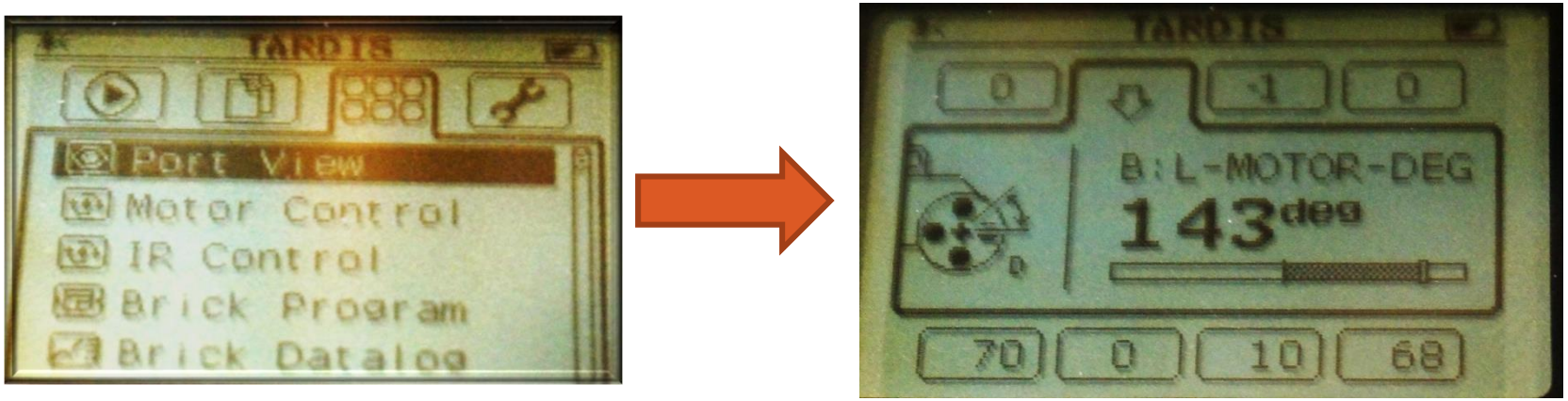
Medium Motor Block



Large Motor Block



USEFUL TIP: PORT VIEW



Rookie Tip: Use Port View on your brick to read sensor values, and measure distances

WHY USE SENSORS IN FLL?

1. **Sensors can help you know your position on the FLL table**
2. **Sensors can help you be more accurate therefore receiving less touch penalties**
3. **Sensors can help you do accurate turns, straighten up on lines, move until a certain distance from a wall, and know when you are on a wall**
4. **Moving up to a mission model accurately might need a sensor**

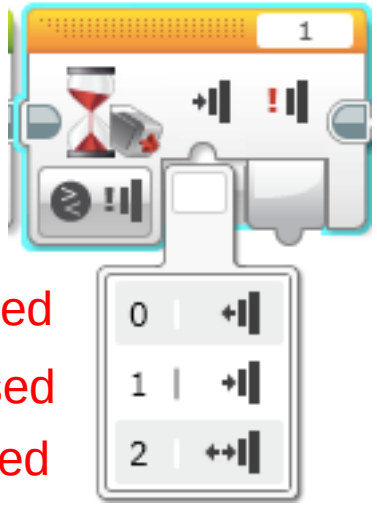
WHAT IS A SENSOR?

- **A sensor lets an EV3 program measure and collect data about is surroundings**
- **The EV3 sensors for FLL include:**
  - Color – measures color and darkness
  - Gyro – measures rotation of robot
  - Ultrasonic – measures distance to nearby surfaces
  - Touch – measures contact with surface

Image from: http://www.ucalgary.ca/IOSTEM/files/IOSTEM/media_crop/44/public/sensors.jpg

WHAT IS A TOUCH SENSOR?

- Touch Sensor can detect when the sensor's red button has been pressed or released
- With this information, you can program an action when the sensor is:



Currently Pressed

Currently Released

Bumped

WHEN MIGHT YOU USE THIS SENSOR IN FLL?

- **Useful for programming “moving until touch sensor is pressed/released”**
- **For example, if you put a touch sensor on the front the robot, you can have it stop moving if it runs into something.**
- **You can also have your program start or stop when a touch sensor is pressed.**

MOVE ON AND OFF

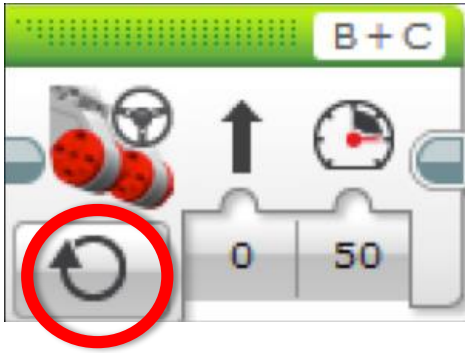
What would happen if you placed a Move Steering Block and left the motor "On"?

Would the robot…

1) Move?

2) Move for a little while?

3) Not move at all?



ANS.

What does Motor Off do?

Rookie Tip: Motor On needs to be followed by another block (e.g. Wait Block)

HOW DO YOU PROGRAM WITH THE TOUCH SENSOR?

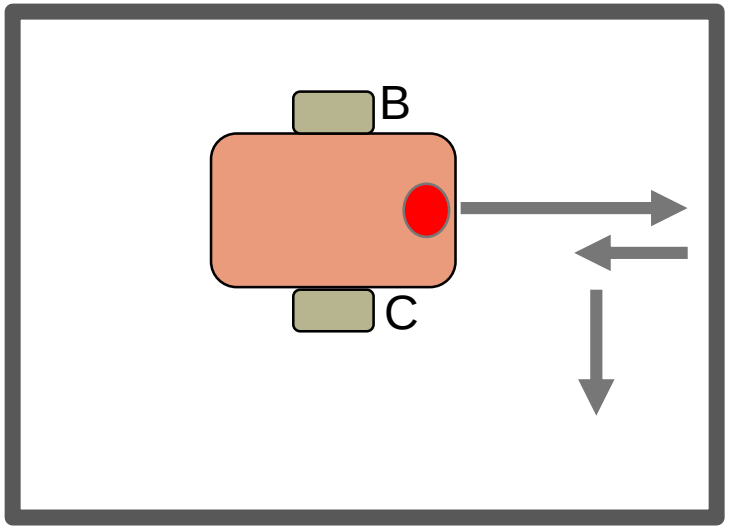
Wait For Block

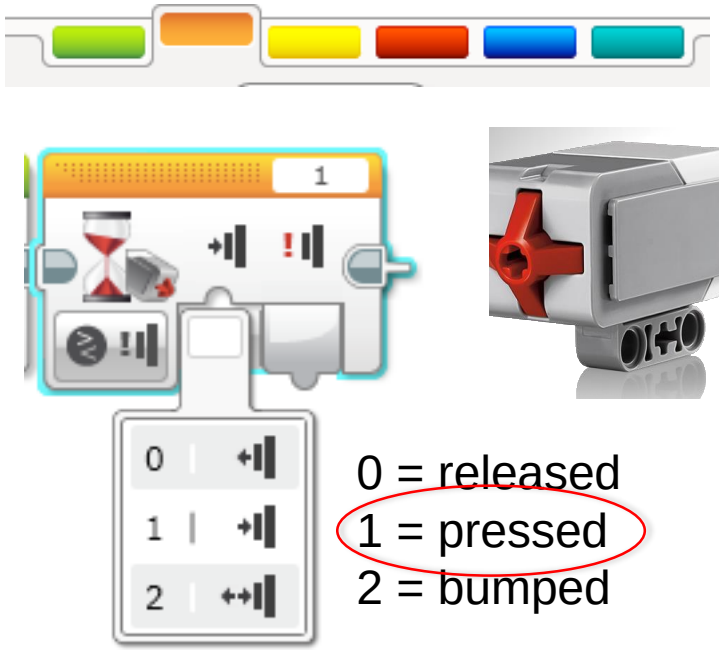
Orange Flow Tab:
Wait for Block

- Used to wait for a sensor reading (or time)

CHALLENGE

Program your robot to move until it hits the edge of a wall. Then back up and turn right 90 degrees.

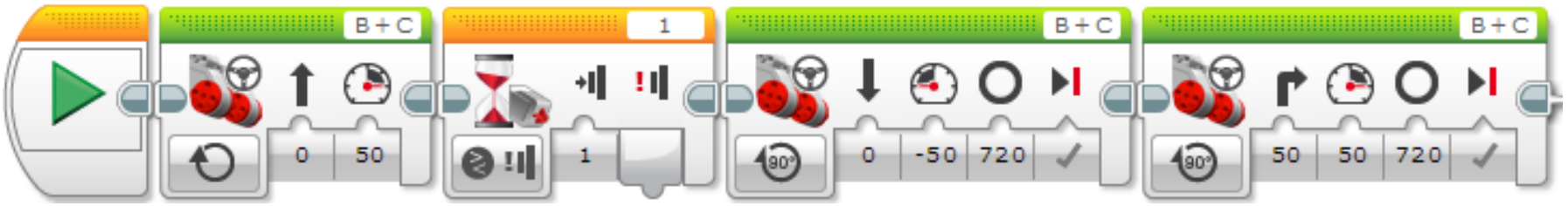




Hint: You will combine Move Steering + Turning + Wait Block

CHALLENGE SOLUTION

The goal of this program is to make your robot move until it hits the edge of a wall. Then back up and turn right 90 degrees



Set move steering block to "on"

Set wail block to Touch-->Compare-->State

Set move steering block to "degrees" and steering to 50. The 720 degrees value will have to be modified for your robot (You measured this in port view earlier beginner lessons).

HOW DO YOU PROGRAM WITH THE ULTRASONIC?

Very similar to the touch sensor

Just change the wait for block to wait for a reading from the ultrasonic sensor

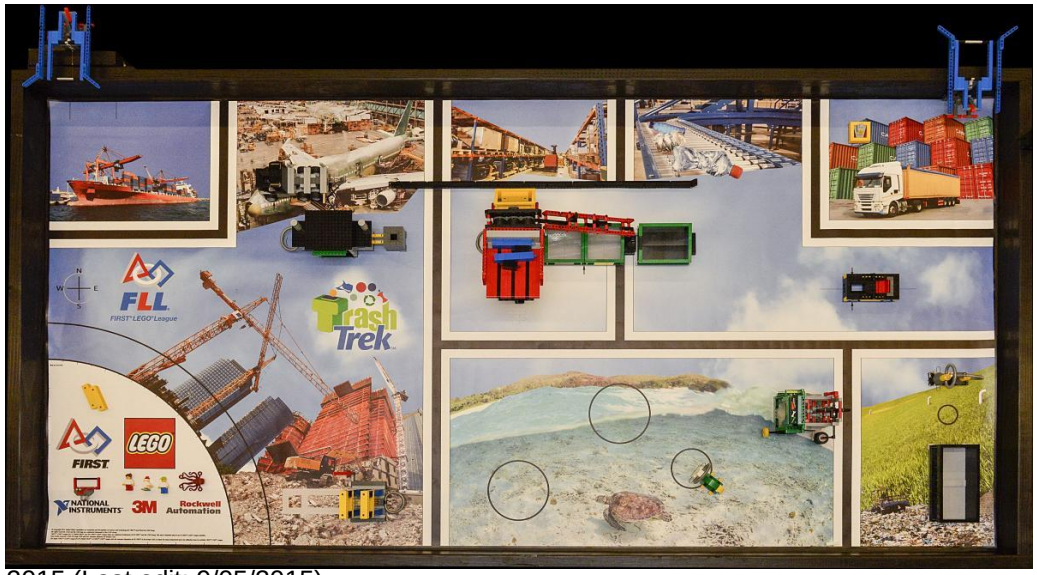
Use it to stay a particular distance away from the wall or mission model.

WHAT IS THE COLOR SENSOR?

- What are they? Sensors that detect the intensity of light that enters it
- Three modes: Color, Reflected Light Intensity and Ambient Light Intensity
  - **Color Mode:** Recognizes 7 colors (black, brown, blue, green, yellow, red, white) and No Color
  - **Reflected Light:** Measures the intensity of the light reflected back from a lamp that emits a red light. (0=very dark and 100=very light)
  - **Ambient Light:** Measures the strength of the light that enters the sensor from the environment. (0=very dark and 100=very light)

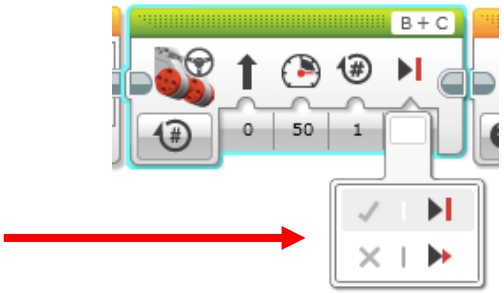
WHY USE A COLOR SENSOR IN FLL?

1. FLL mats have many lines that you can use to make your robot more reliable
2. Use the color sensor to move until a line, follow a line, and square up on a line
3. We will teach you to move until a line and follow a line in this workshop. The other square up on a line lesson is available on EV3Lessons.com



ANOTHER MOVE STEERING TIP: COAST OR BRAKE?

- Something more about the Move Steering Block
- You will notice you have an option to COAST or BRAKE
- Coast will make the motors keep moving. Brake makes the motors stop immediately.
- Which do you use to stop EXACTLY on a colored line?



COLOR SENSOR CHALLENGE

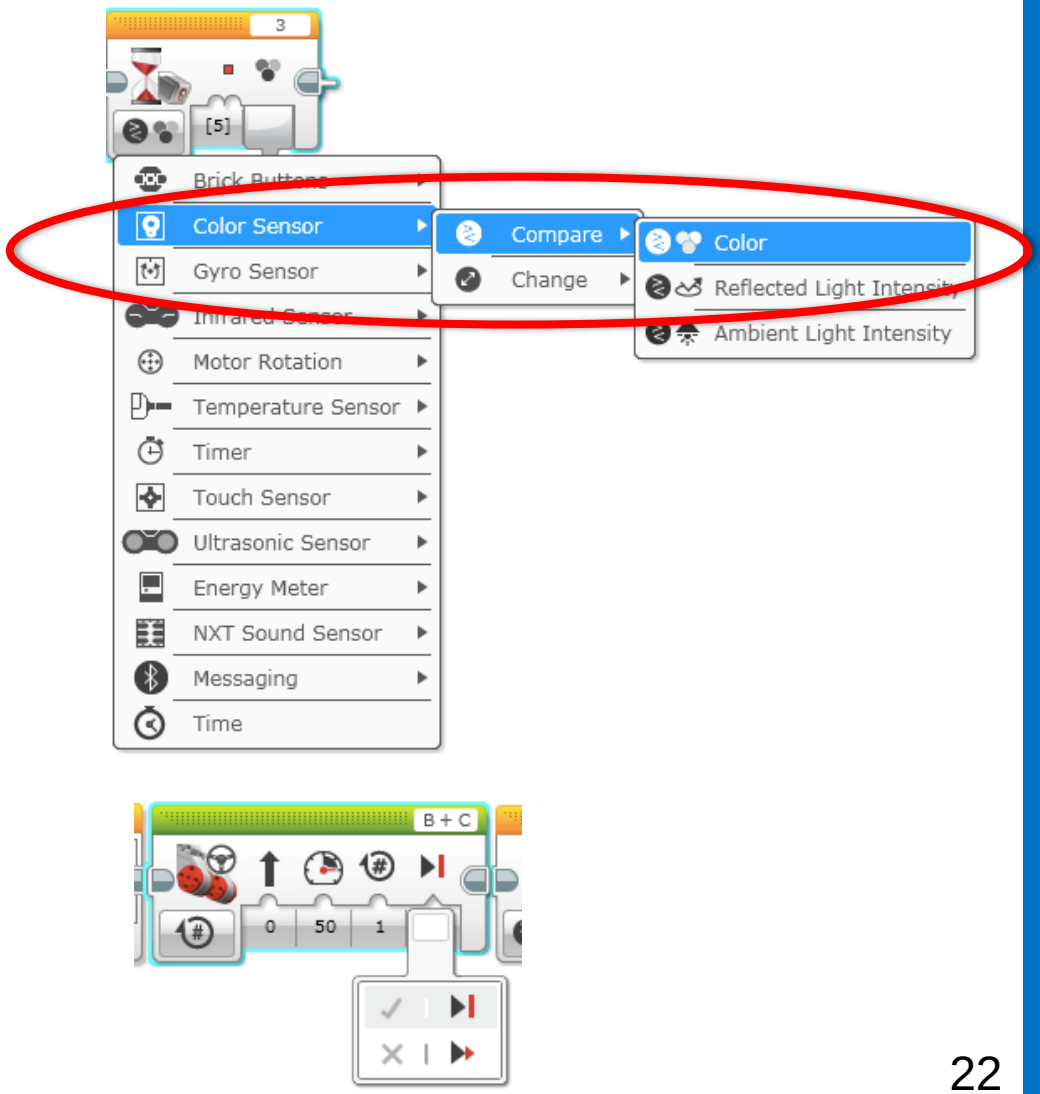
Make the robot move up to a black line using the color sensor?

Step 1: Use Wait For Color

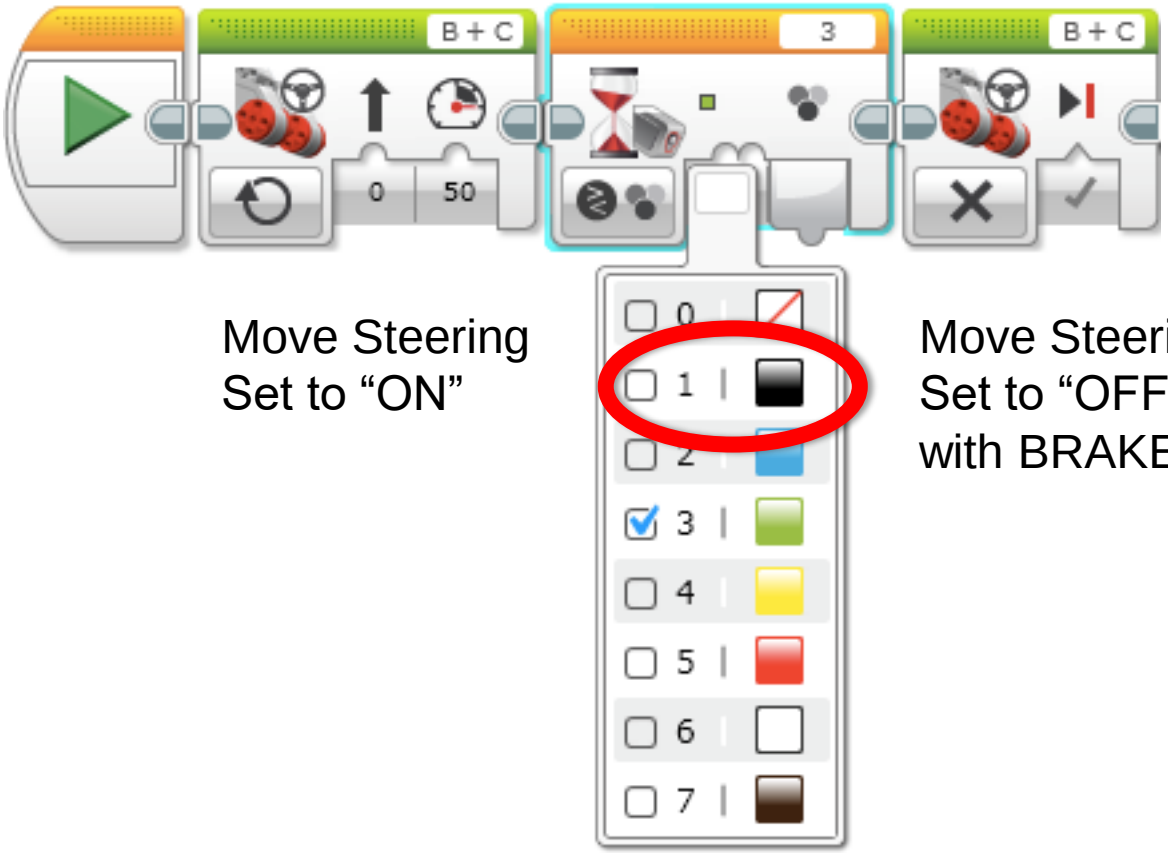
Step 2: Use the color sensor in COLOR MODE

Step 3: Coast or Brake?

Hint: You will use Move Steering (think about motor on and off) and Wait for "Color"



COLOR SENSOR CHALLENGE SOLUTION



Move Steering
Set to “ON”

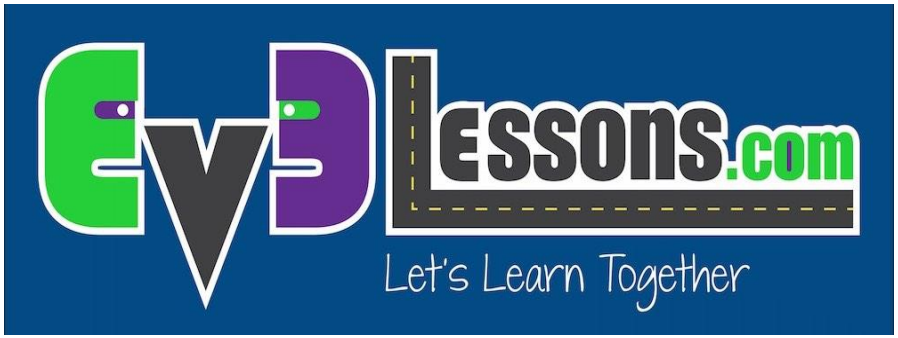
Move Steering
Set to “OFF”
with BRAKE

Wait until Color is Black (#1)

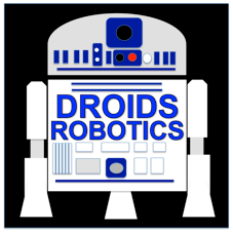
FINISH

START

BEGINNER PROGRAMMING LESSON



Basic Line Follower



By: Droids Robotics

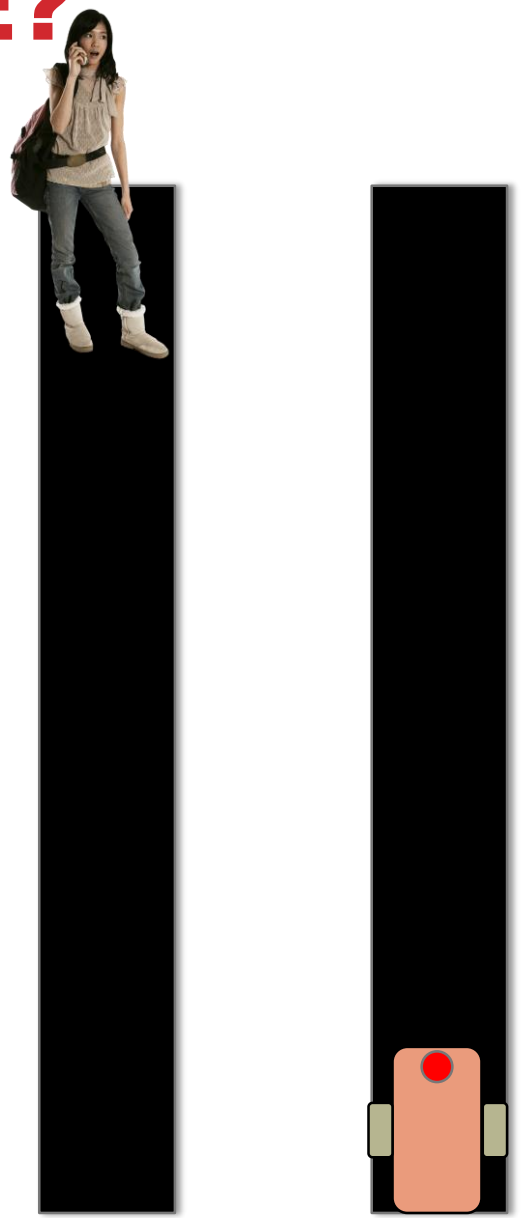
WHERE LINE FOLLOWING COULD BE USEFUL IN FLL

Trash Trek mat is covered with black lines

Lines go up to useful regions

Lines go up to mission models

FOLLOW THE MIDDLE?

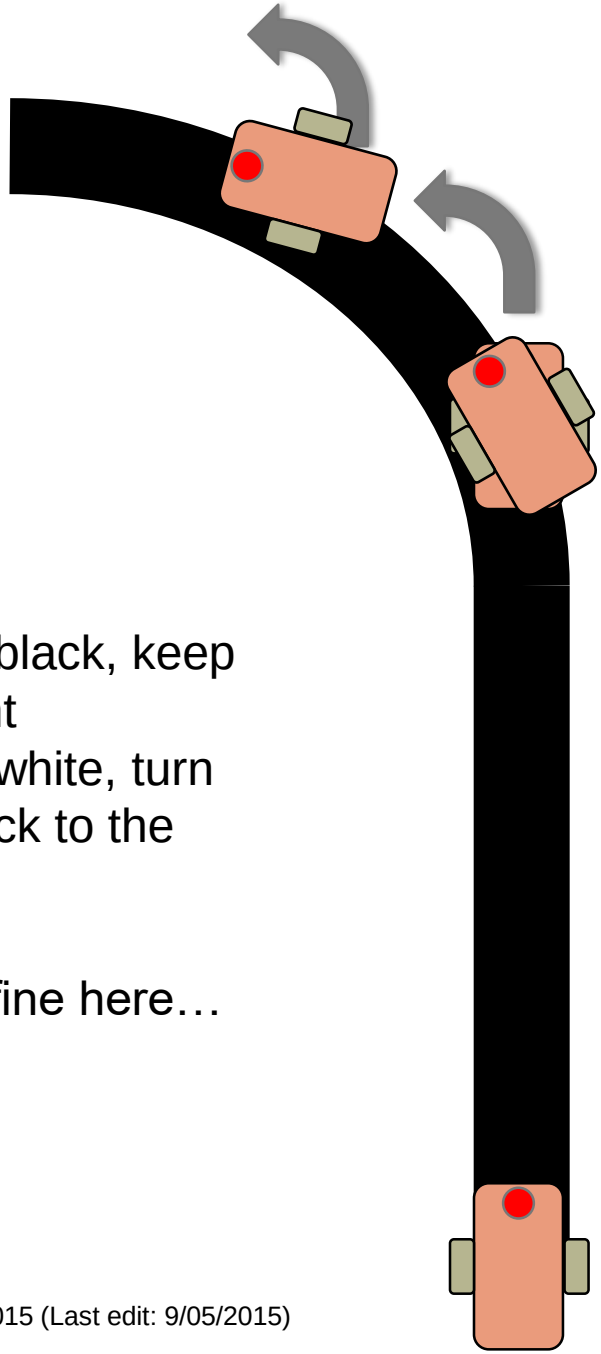


Humans want to follow the line in the middle.

Let's have the robot do the same thing using the Color Sensor

What type of questions can we ask using this sensor

- Are you on line or not?



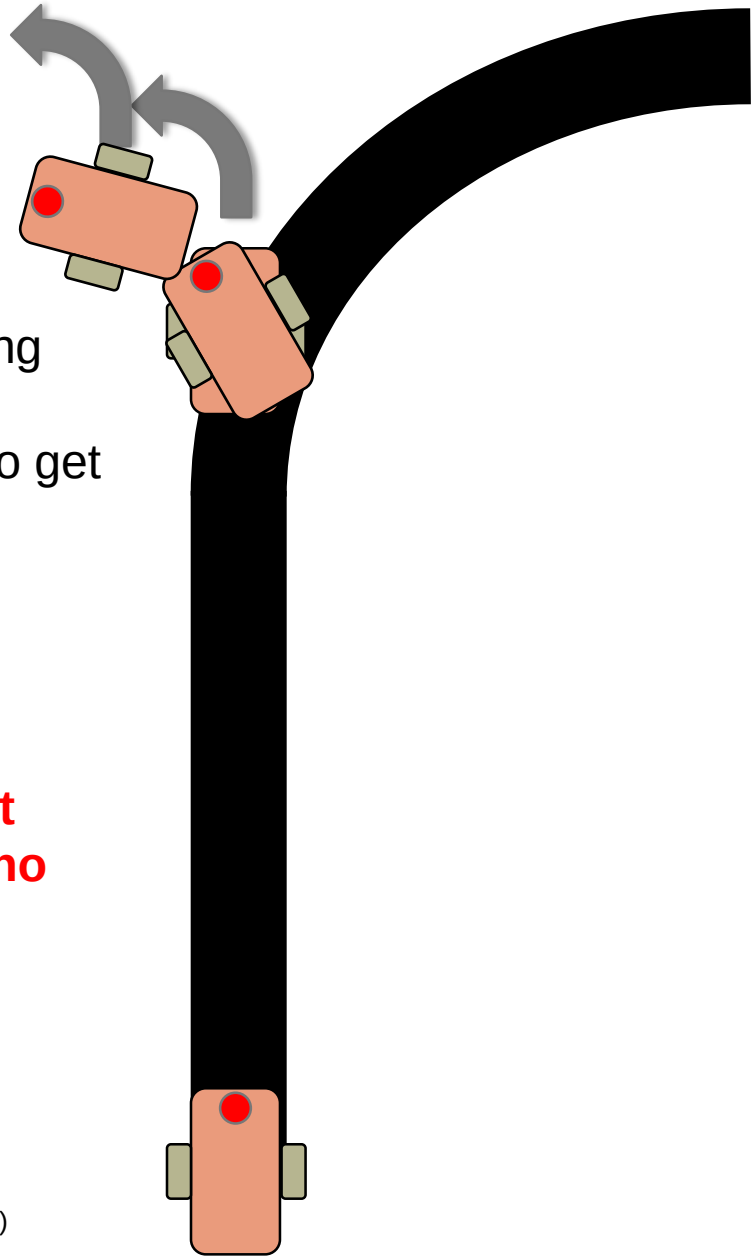
1. If we are on black, keep going straight
2. If we are on white, turn left to get back to the line

Seems to work fine here…

1. If we are on black, keep going straight
2. If we are on white, turn left to get back to the line

OH NO… my robot is running away….

When the robot leaves the left side of the line, the program no longer works!



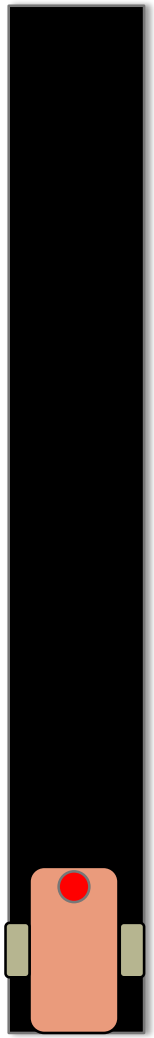
LINE FOLLOWING: ROBOT STYLE

Why could the Human follow the middle?:

- They can see ahead.
- They can see the whole line and its surroundings
- They see both sides and which side they left

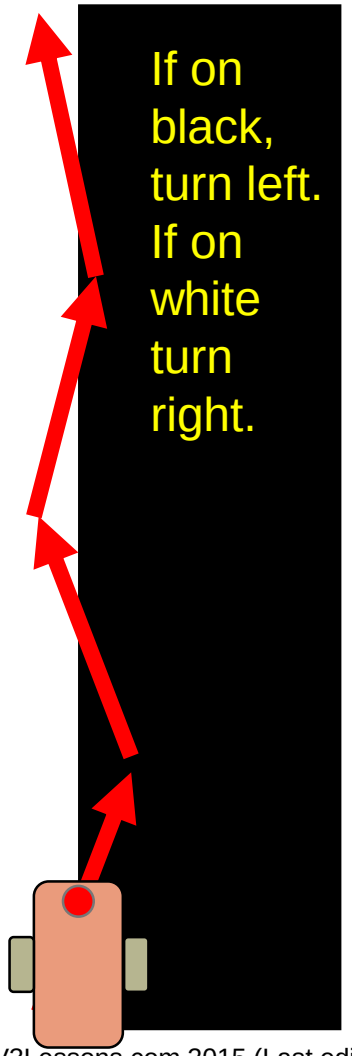
Why can't the Robot do the same thing?:

- Can't tell right or left side of the line
- How do we make sure the robot always veers off on the SAME SIDE of the line?
    - Instead of the middle, could the robot follow the "edge"?
- So now the robot will fall off only the same side.
- We will now show you how this works!



ROBOT LINE FOLLOWING HAPPENS ON THE EDGES

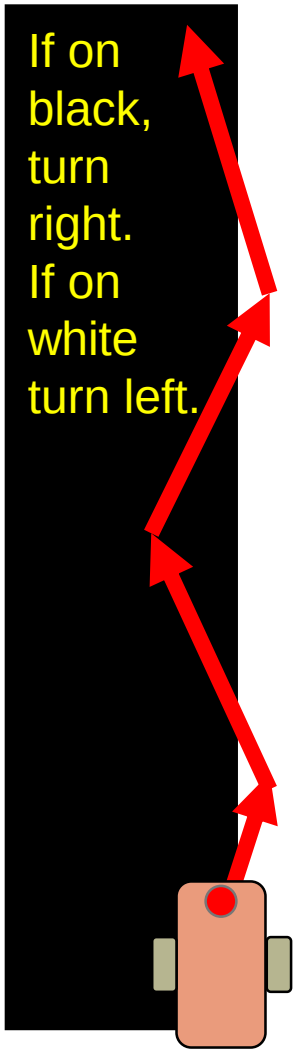
Left side line following



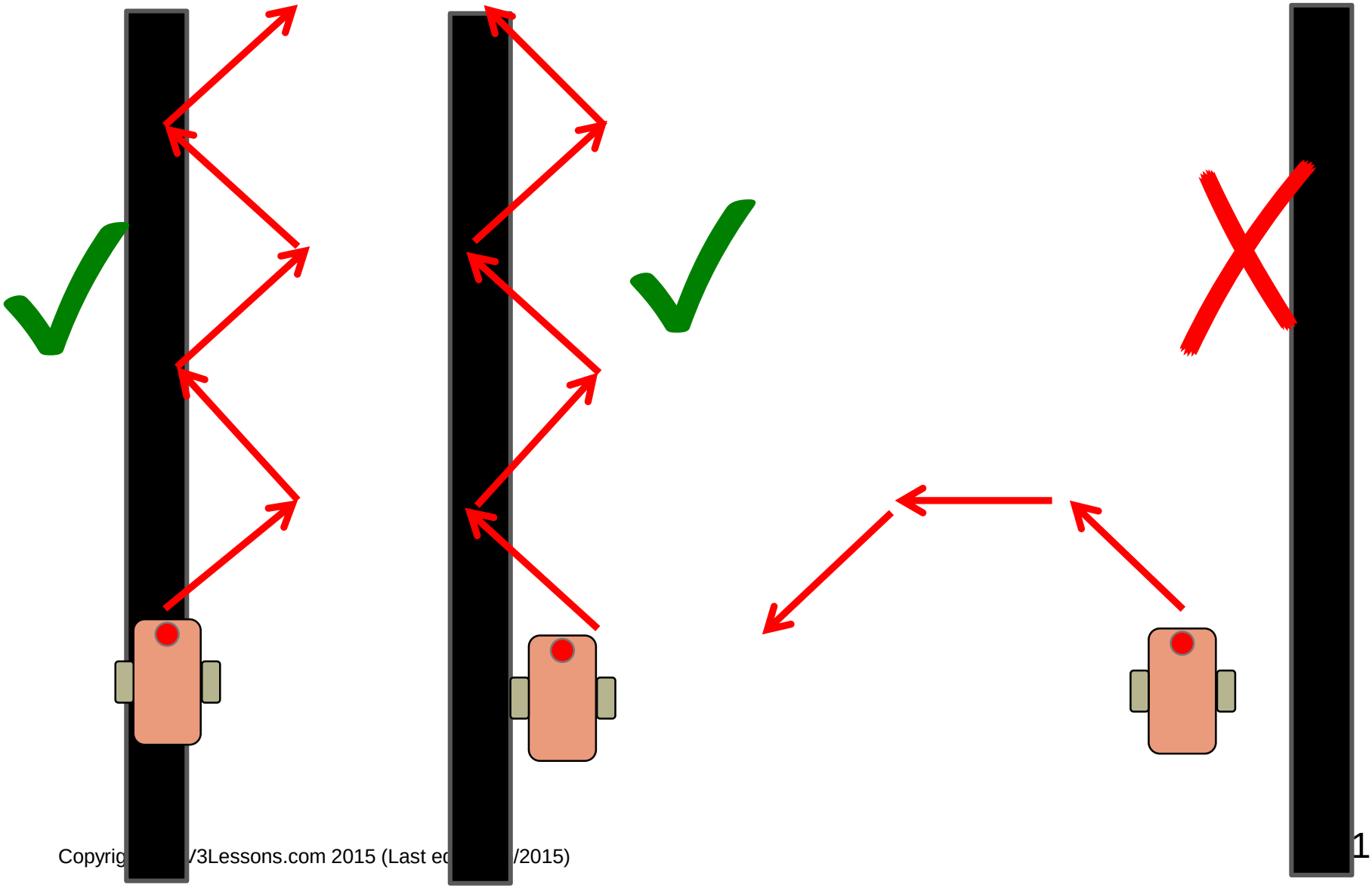
The robot has to choose which way to turn when the color sensor sees a different color.

The answer depends on what side of the line you are following!

Right side line following



STARTING THE ROBOT ON THE CORRECT SIDE



HOW DO YOU WRITE A LINE FOLLOWER?

First, the robot will have to decide between 2 actions. What are the two actions?

ANS:

Second, the robot is repeating an action over and over again. What action does the robot repeat again and again?

ANS:

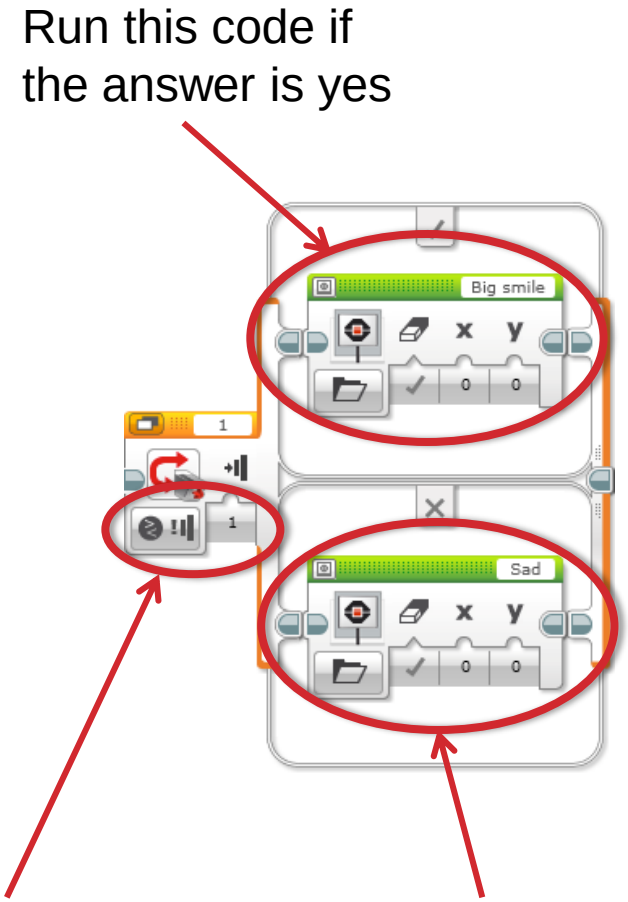
SWITCH BLOCKS

Asking the robot a question and doing something different based on the answer

- Example: Does the robot see a line? Or not?

Basically a YES/NO QUESTION

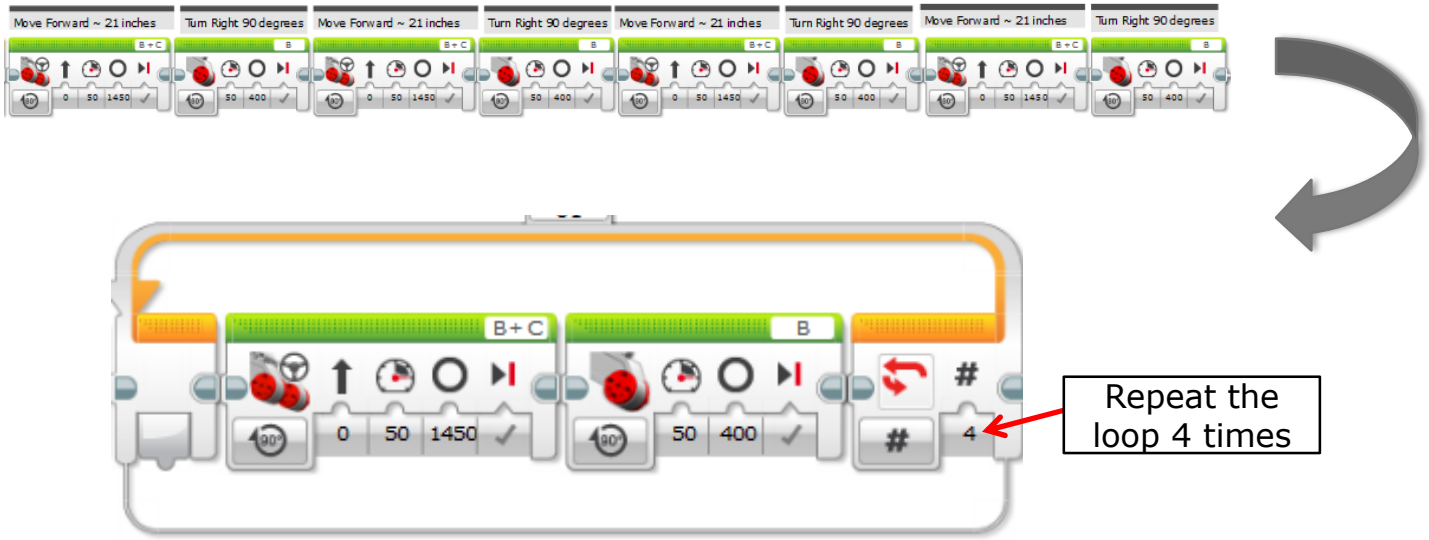
Switch blocks are found in the orange/flow tab



LOOPS

Loops make repeating a task multiple times easy

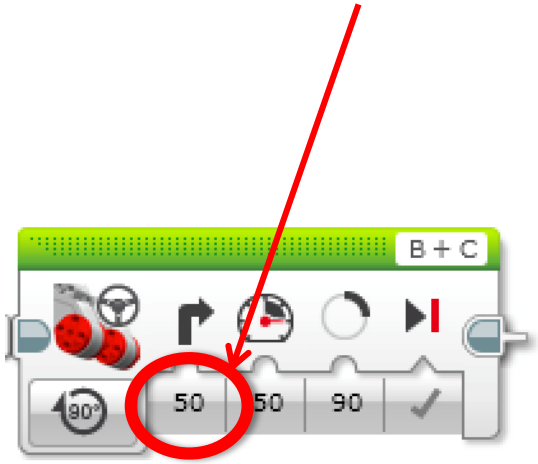
- KEEP GOING….Forever, for a Count, Until touch (or something else)

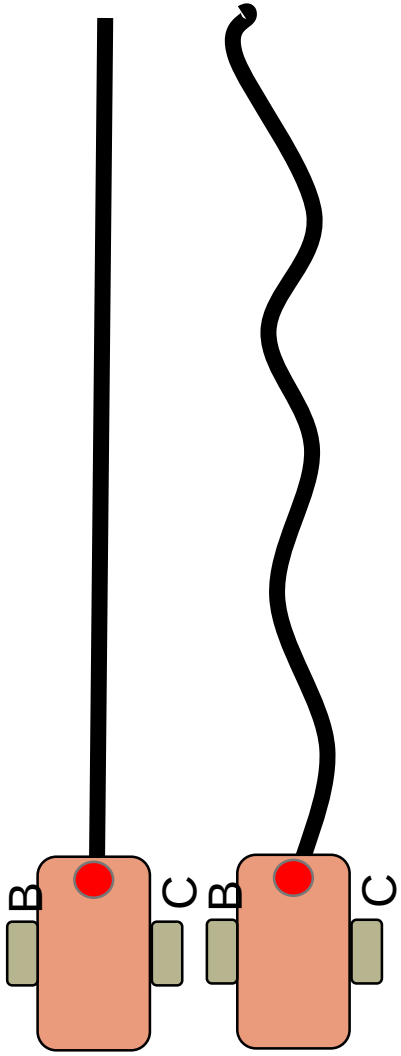


LINE FOLLOWER CHALLENGE 1

Step 1: **Write a program that follows the RIGHT edge of a line.**

Hints: If your sensor sees black, turn right. If your sensor sees white, turn left. Use loops and switches!





OTHER THINGS TO TRY

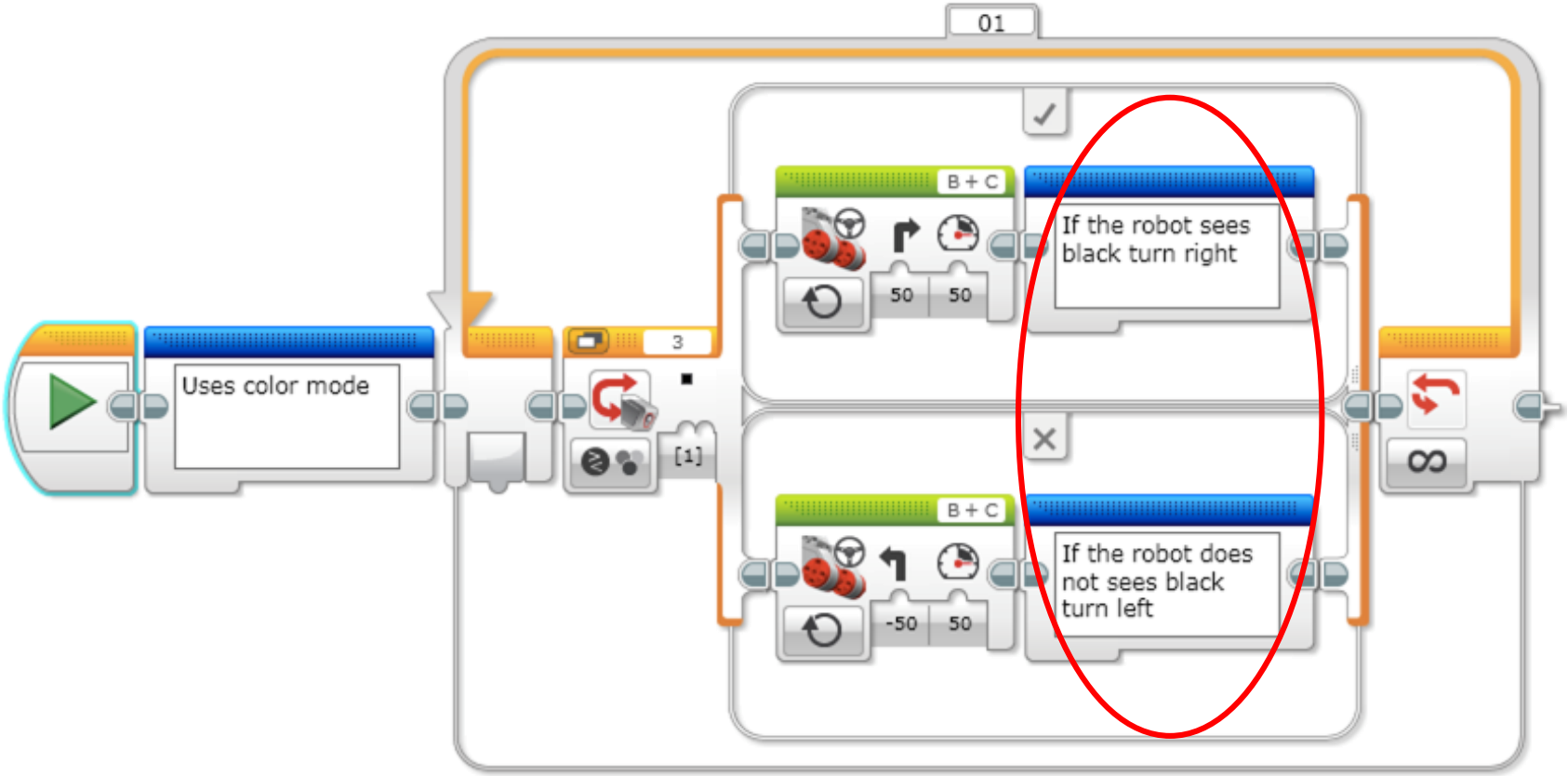
Try the line follower on different lines.

Can you write a smoother line follower (less wiggle)?

Can you have your robot line follow for a particular distance?
(Hint: Use Port View to measure the distance)

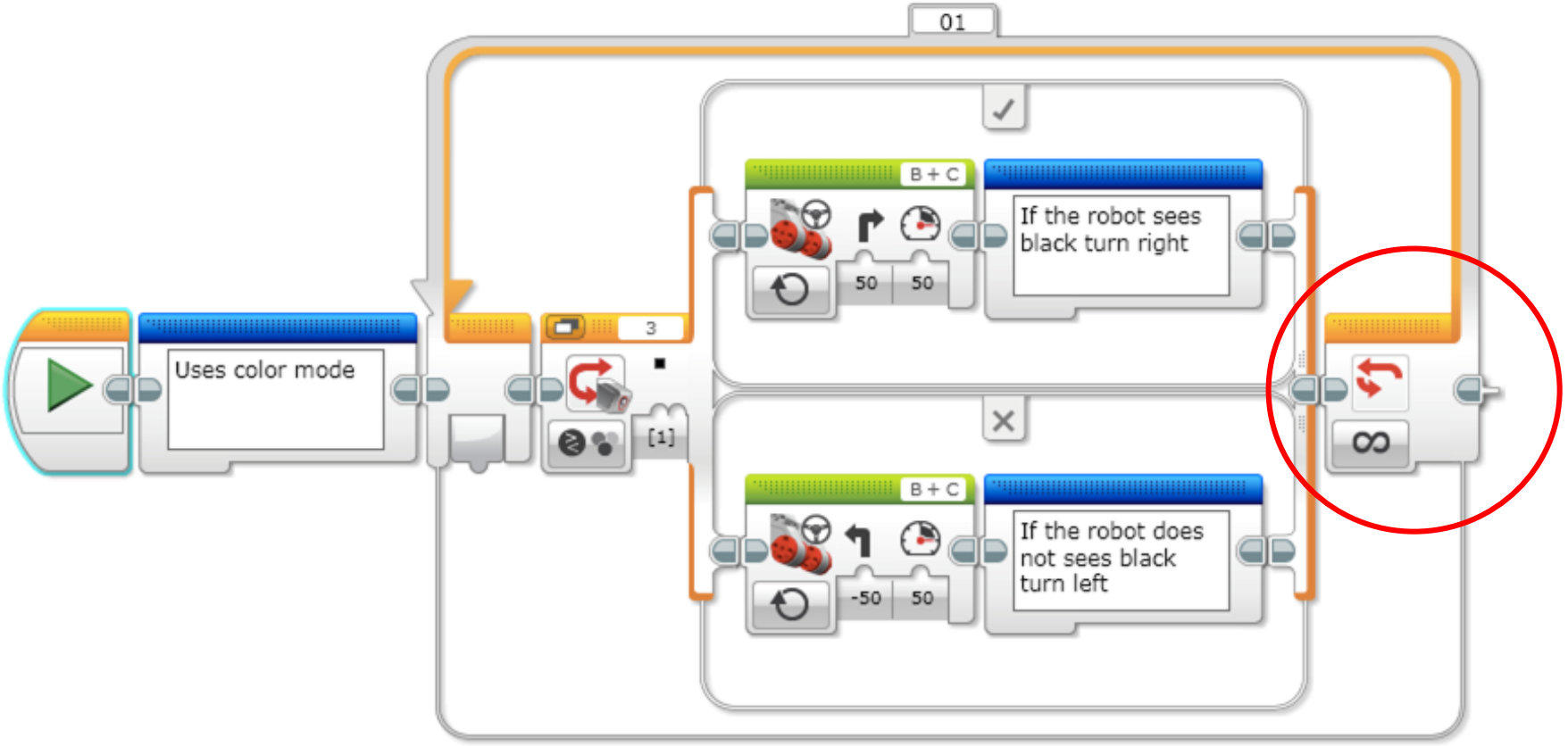
Can you have your robot line follow until it finds an intersection? (Hint: Use the second color sensor)

LINE FOLLOWING CHALLENGE SOLUTION



Q. Does this program follow the Right or Left side of a line?
A. The robot is following the Right Side of the line.

CHALLENGE 1 SOLUTION



Q. This line follower goes forever. How do we make this stop?
A. Change the end condition on the loop.

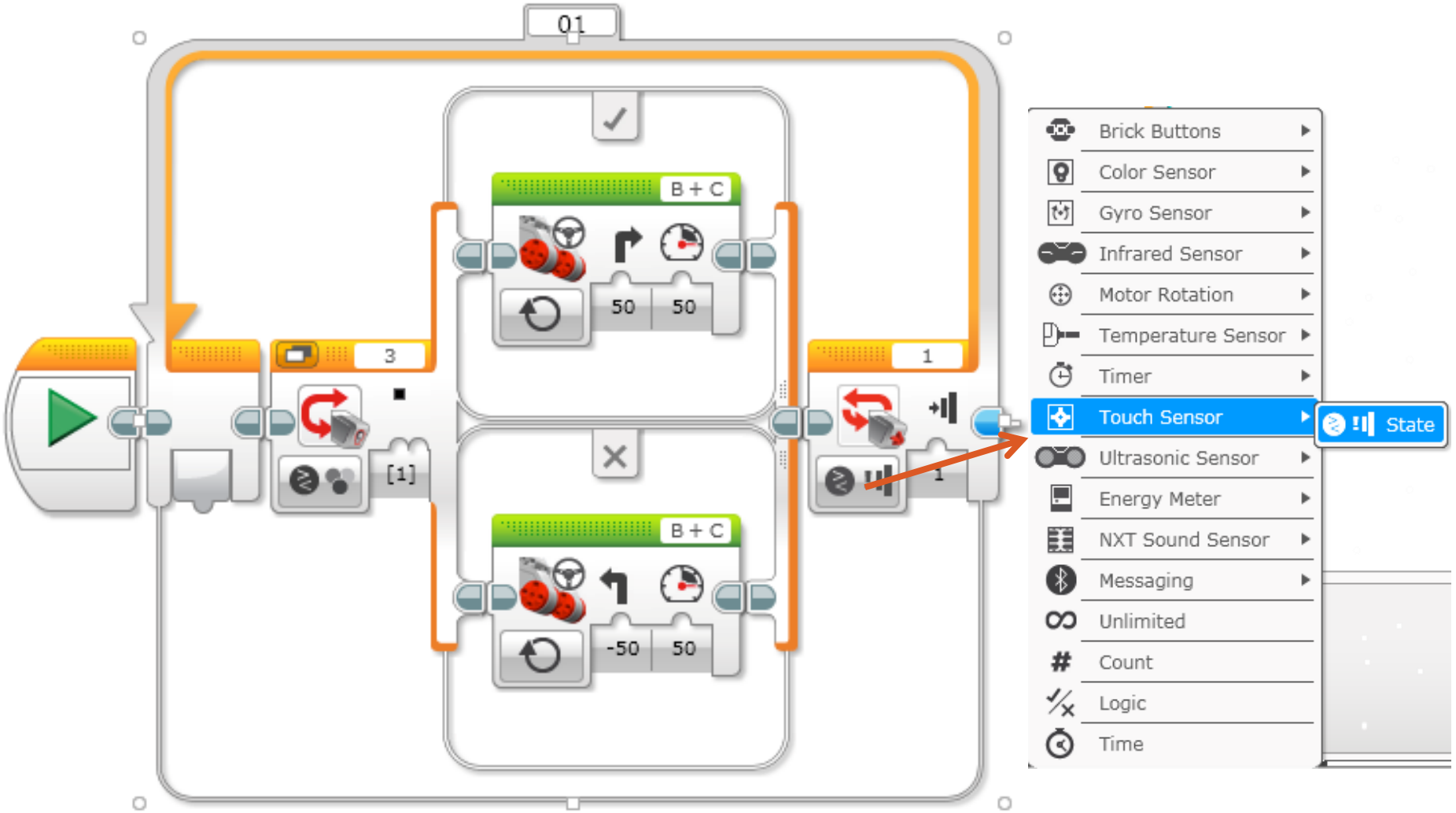
LINE FOLLOWER CHALLENGE 2

Part 1: Make a line follower that stops when you press the touch sensor

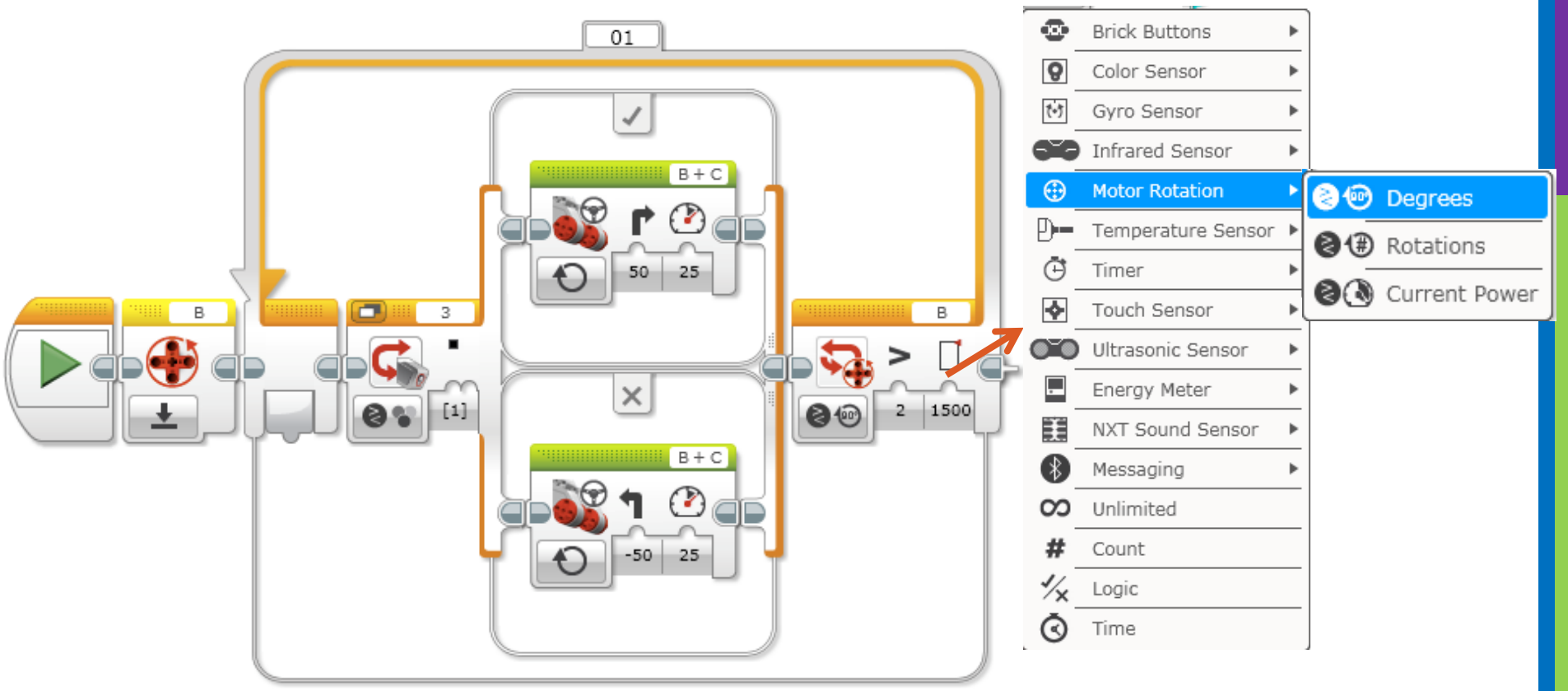
Part 2: Make a line follower that stops after it travels a particular distance

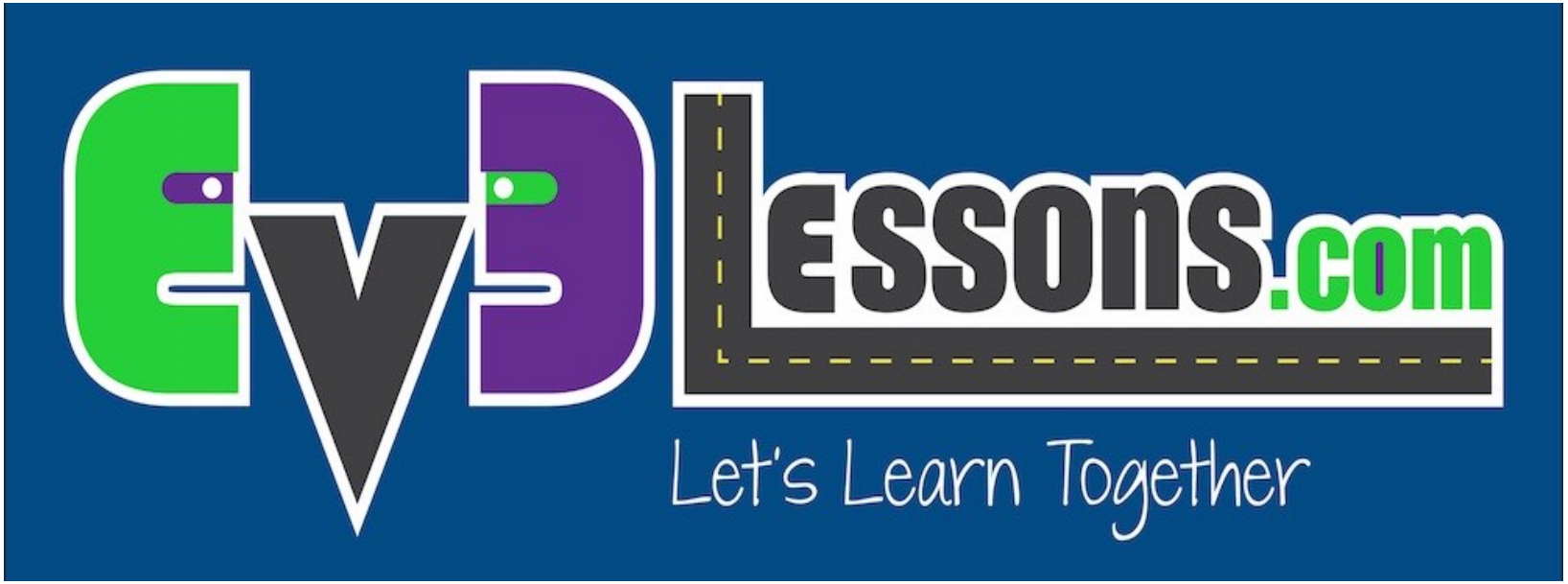
Hint: You will have to use PORT VIEW in your brick to measure the distance first

CHALLENGE 2 SOLUTION: SENSOR



CHALLENGE 2 SOLUTION: PARTICULAR DISTANCE





Branching Error
(a.k.a. the VM Program Instruction Break Error)

By Sanjay and Arvind Seshan

DEBUGGING LESSON

HISTORY

- We first encountered the “VM Program Instruction Break” error on our brick during the fall of 2013 during the Nature’s Fury FLL season. We searched online for any documentation about this error, but could not find any. We were the first to report this problem on the FLL Forum.
- Many FLL and WRO teams have encountered this error since then. While they persisted and tried to come up with a work-around, the solution was never enough.
- Without knowing what was causing the error, it was difficult to come up with a permanent solution. The only solution available at the time was trial and error.
- This presentation documents what the underlying causes were and the solution.

COMMON SYMPTOMS

- **The robot stops in the middle of a program and displays “VM Program Instruction Break” on the screen**
- **Adding debugging code may make the error appear in a different location of the code.**
- **The error would appear even with minimal changes to the code such as the movement of the relative position of two My Blocks**
- **Often occurs in more complex programs (e.g. it often happened to us each season as we added more code to our main sequencer)**

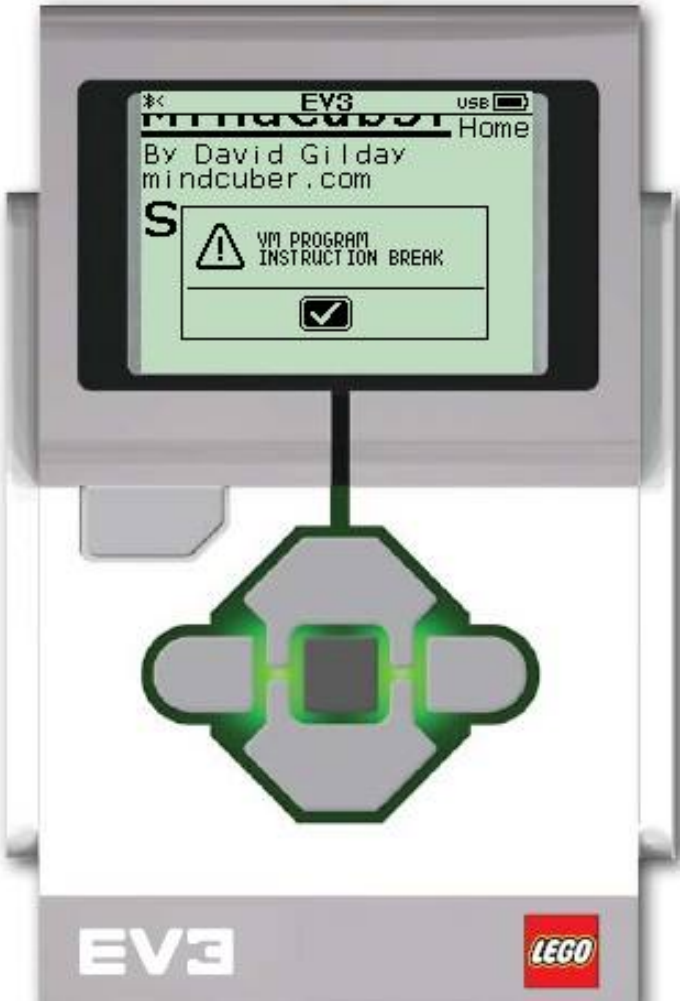


Image provided by David Gilday

WHAT IS A VM?

A virtual machine (VM) is an emulation of a computer system. This "emulated" system maybe totally different than the computer you run the VM on. For example, you may run a VM emulating an iPhone on your laptop to run or test phone software.

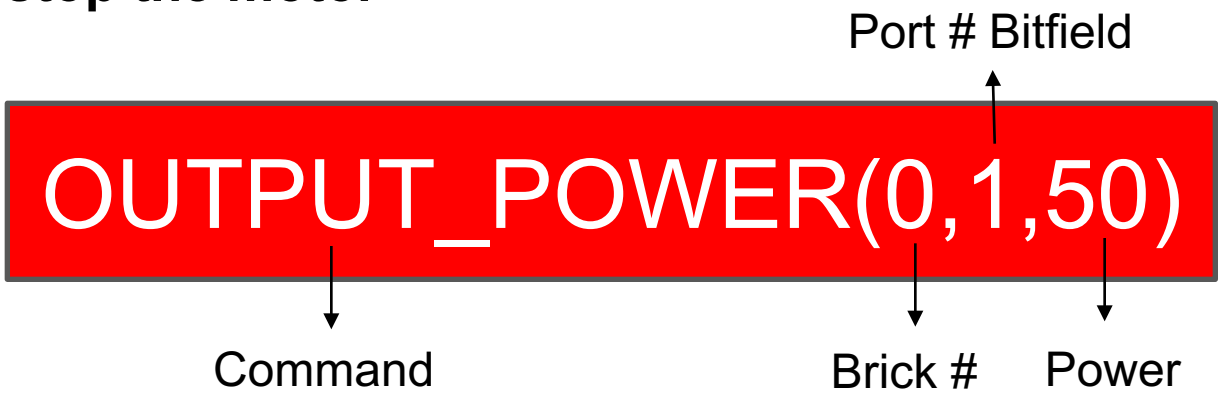
The EV3 uses a TI's Sitara AM1808 ARM9™ processor running the Linux OS. However, the code you download to the EV3 is not a ARM9 binary. It contains EV3 "bytecode" that is interpreted by a VM running on the EV3.

The bytecode for the EV3 defines a simple set of instructions to perform computations and access the hardware connected to the EV3 (screen, bluetooth, motors, etc.)

WHAT IS BYTECODE?

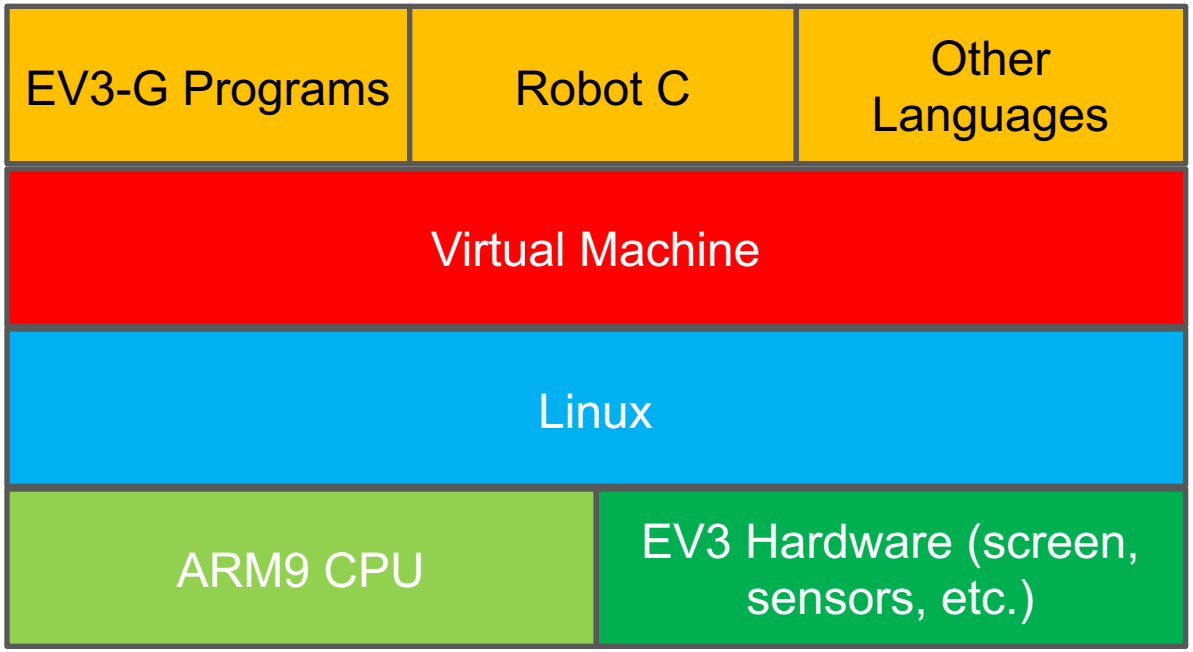
The bytecodes are closely related to the blocks you see in EV3-G. For example:

BYTECODE: OUTPUT_POWER(0,1,50). This particular command sets the power of the motor on port 1. Other bytecodes actually start and stop the motor



To learn more, visit:
http://analyticphysics.com/Diversions/Assembly%20Language%20Programming%20for%20LEGO%20Mindstorms%20EV3.htm

WHAT ROLE DOES THE VM PLAY



The VM sits between your programs and the operating system running on the EV3

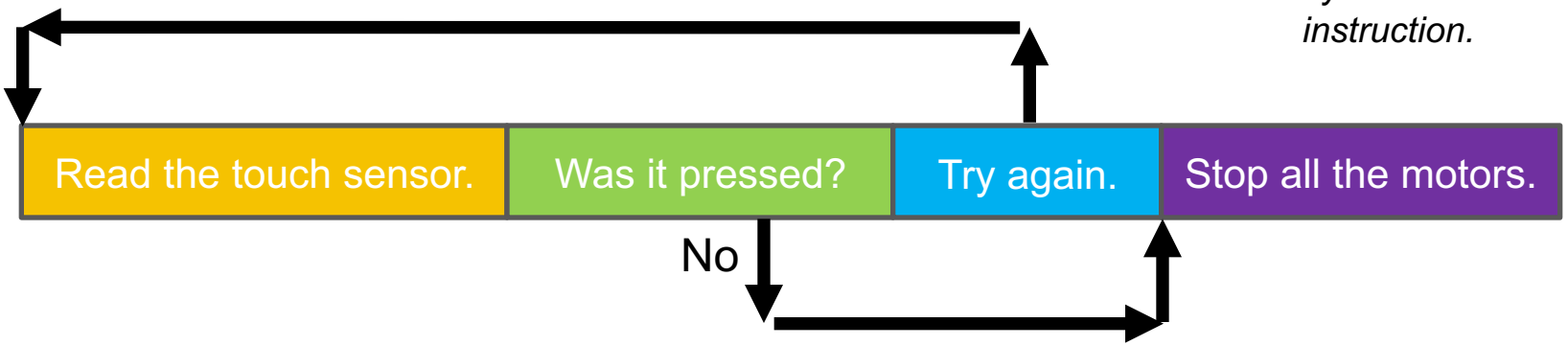
Note that systems such ev3dev run their own updated Linux installation with their own drivers for the EV3 hardware (i.e. they don't use a VM bytecode interpreter)

SOURCE OF THE PROBLEM

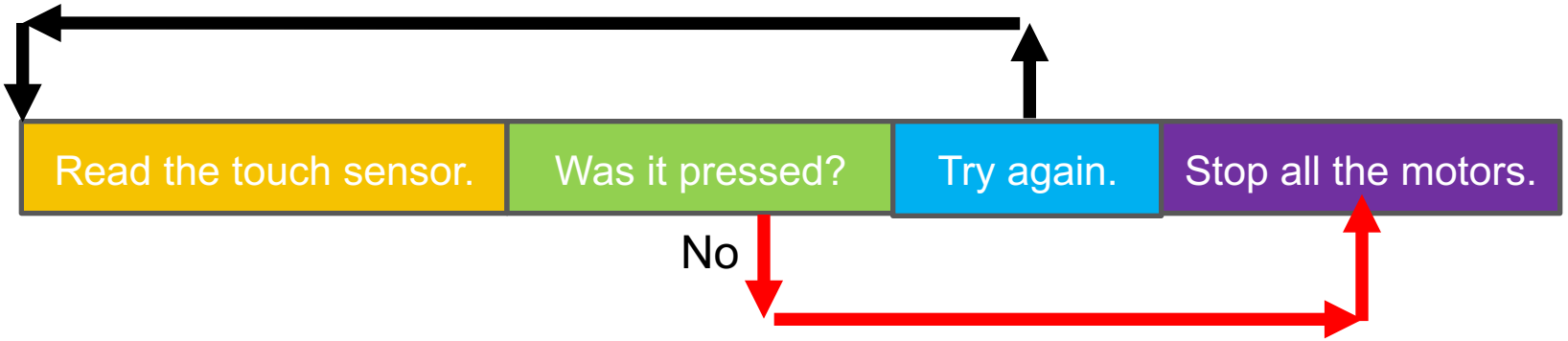
- **Was it really a bug in the VM?**
  - No. Turns out it an issue with the compiler on your PC generating incorrect bytecode. Specifically, it was a problem with the branches in the code generated.
- **What is branch code?**
  - Normally, your EV3 executes instructions in sequential order
  - A branch (or jump) instruction is one that tests a condition (e.g. is the button pressed) and causes the EV3 to jump to a different set of instructions if the condition is met
  - Branches are used to implement Switches, Loops and almost any command that results in different possible results.
  - EV3 bytecode has unconditional branches that always jump to another piece of code, and conditional branches that test one or two pieces of data

A SIMPLE VIEW

Each box is one bytecode instruction.



What you want your code to do: In the top case, the branches jump to the beginning of each sentence

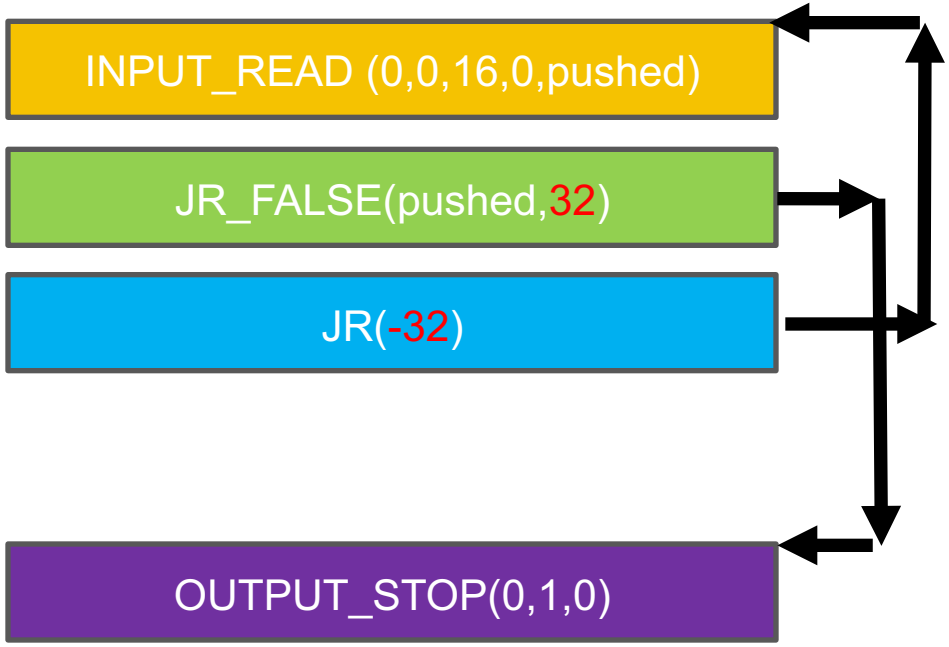


What happens in a VM Program Instruction Break: In the bottom case, the branch jumps too far. The EV3 tries to interpret what the command "the motors" means and fails.

A BYTECODE VIEW

buttonPushed:	This is the “button pushed” loop label
INPUT_READ (0,0,16,0,pushed)	Read touch on port 1 in touch mode and store in variable “pushed”
JR_FALSE(pushed,buttonNotPushed)	If “pushed” is FALSE exit the loop by jumping to the buttonNotPushed code. If it is not pushed, it just goes to the next instruction
JR(buttonPushed)	Go back to the beginning of the “button pushed” loop label
buttonNotPushed:	buttonNotPushed: label
OUTPUT_STOP(0,1,0)	Stop motor B

EXAMPLE BYTECODE



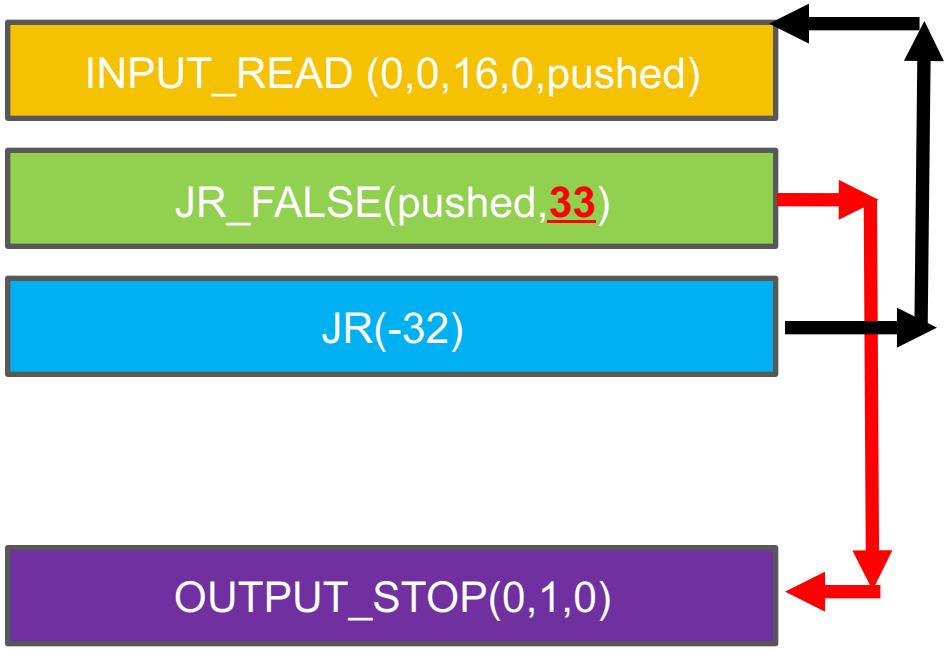
Actual executed code does not include the labels, but does include the offsets.

Length (or offset) of the jump is in red.

Arrows point to the destination of the jump.

Note, the jump is to the start of each command.

THE PROBLEM



The offset of the branch was sometimes calculated incorrectly. In this case, it says "33" instead of 32 (in red).

As a result, the branch would jump to the middle of OUTPUT_STOP instruction. This is like jumping to the middle of a sentence. Most often the partial instruction made no sense and the VM would respond with a "VM Instruction Break"

Sometimes the partial command was a valid instruction – just not the one you wanted. Therefore, your robot would act incorrectly.

WHY? AND WHAT HAPPENS NOW?

- **The source of the problem is that the code compiler on your PC calculated an incorrect branch length (offset).**
- **LEGO has released an update to the EV3 programming software with a bug fix**
  - As of of 10/25/2016, both Retail and Education editions of V. 1.2.2 are available for download
- **Download and install the update on your PC**
  - After that, you can load any program you had symptoms such as “VM Instruction Break” that were caused by the bad branches and just download again to your EV3. The newly downloaded code should not have any bad branching code!

SOME LESSONS

- **Reporting errors can be useful**
  - A big part of finding the solution to the “VM Program Instruction Break” error was FLL, WRO teams and other robot builders reporting and discussing errors
  - Similar to when you see a Google or Microsoft “report this error” message on your screen.

- **Learning debugging skills**
  - FIRST LEGO League teams, in particular, faced this error as their code became come complex
  - They persisted and worked around the problem as well as they could.

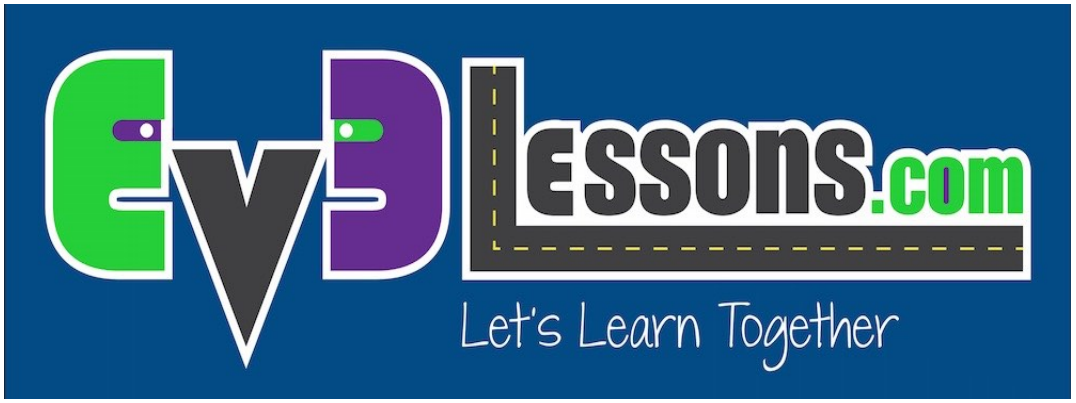
A COMMUNITY EFFORT

Thank you to MINDSTORMS Community Partners, FLL Teams, WRO Teams, other builders in the community, National Instruments, and LEGO who worked together to identify this error and create a solution.

CREDITS

- **This tutorial was created by Sanjay Seshan and Arvind Seshan**
- **More lessons at www.ev3lessons.com**

INTERMEDIATE PROGRAMMING LESSON



BRICK BUTTONS AS SENSORS

By Sanjay and Arvind Seshan

SESHAN BROTHERS

Lesson Objectives

Learn how to use your brick buttons as sensors

Prerequisites: Display Blocks

What are the Brick Buttons?

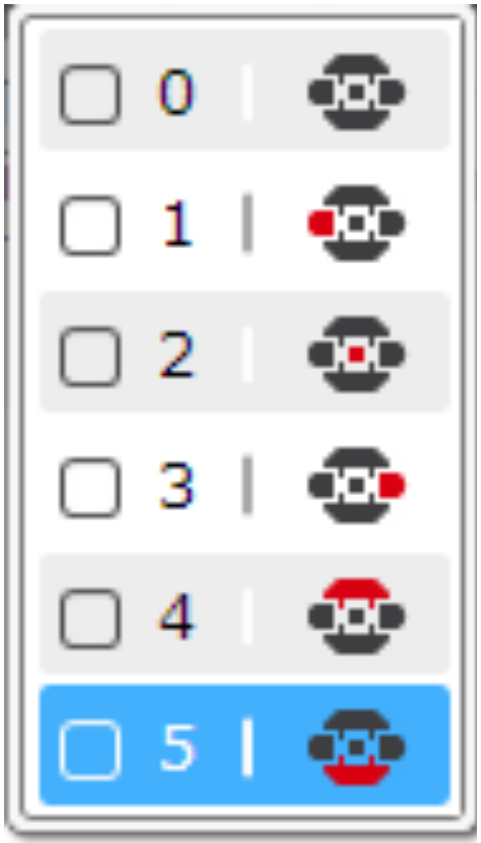
The Brick Buttons are the five buttons on the EV3 Brick (all buttons except the Back button)

They can be used as sensors to detect if a button has been pressed, to find out which button was pressed and to control your program

You can even use them to track if the button was pressed and then released in the past (like Bumped for the Touch Sensor)

Note: You cannot detect if two buttons are pressed at the same time

Wait Blocks, Switches, Loops and the Brick Buttons Programming Blocks all let you use the brick buttons as sensors



Challenge 1: Button Press & Debugging

CHALLENGE: Program your robot to move forward until a button is pressed. Just like in the beginner lessons that used sensors, you will use a Wait For block to complete this challenge.

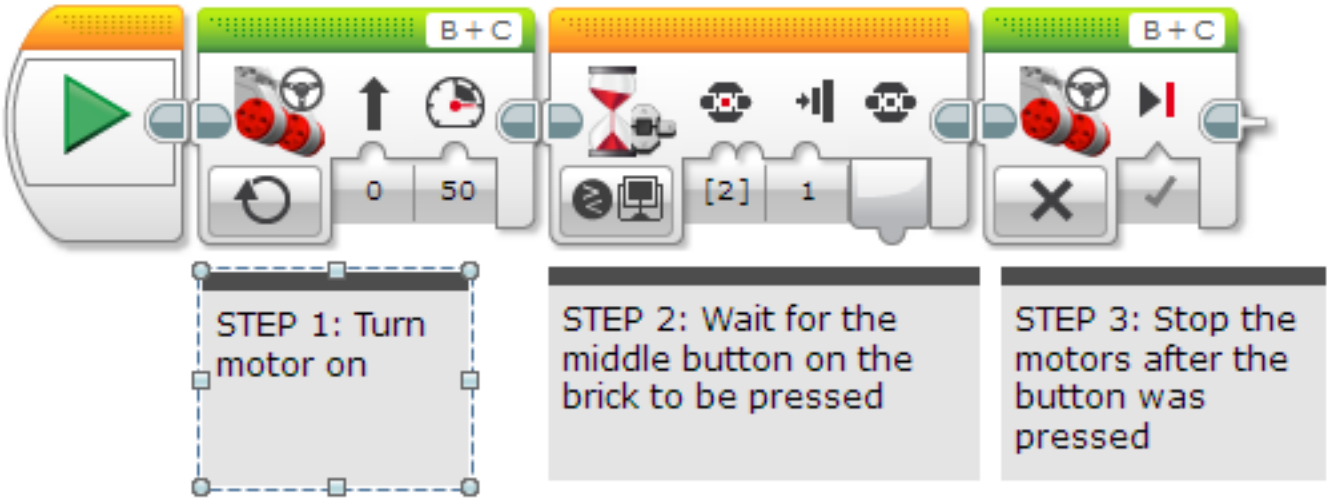
A good use for such a program might be to help you debug. Adding a Wait For Button Press in your code helps you run parts of your code and check for errors.

STEP 1: Turn on motor in your Move Steering Block

STEP 2: Add a Wait For Button Press (Middle Button)

STEP 3: Stop Motors after the button is pressed.

Challenge 1 Solution



Challenge 2: Menu With Buttons

Create an onscreen menu using the brick buttons that does a different action based on which button is pressed. The actions to program are – go forward, backward, left and right

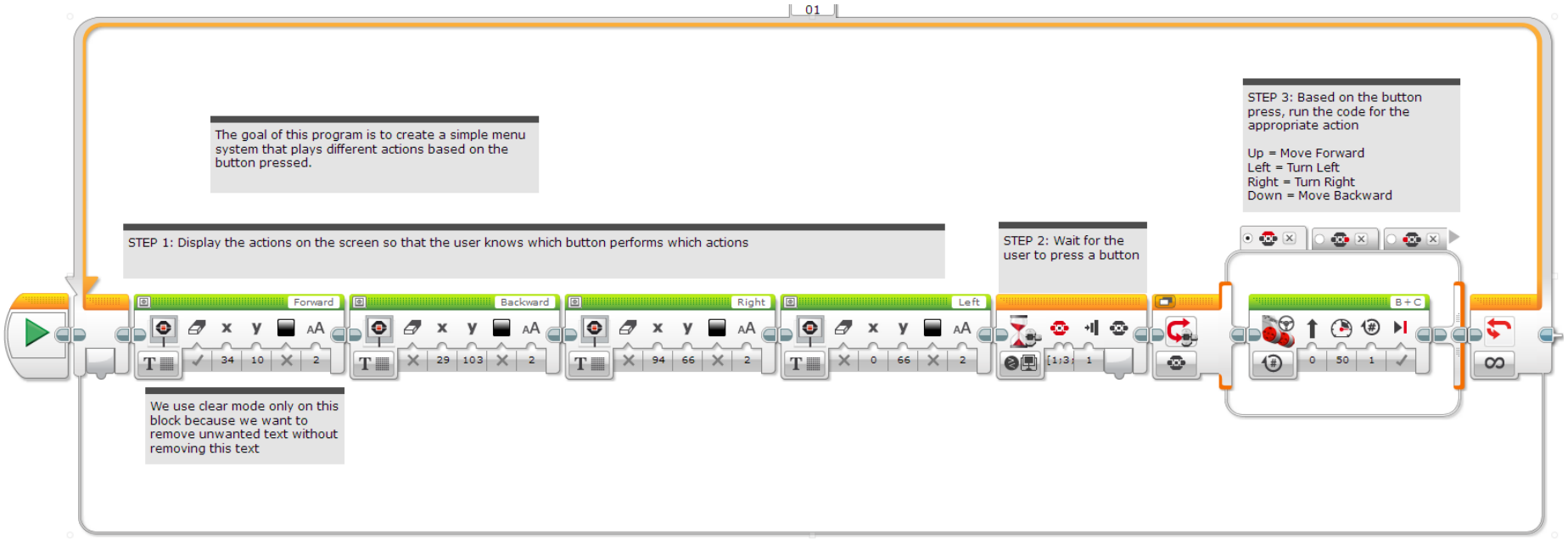
STEP 1: Use four Display Blocks to display the 4 actions on the screen so it will look like the image on the right

STEP 2: Add a Wait For button press

STEP 3: Add a Switch Block and based on which button is pressed, run the code for the four actions – left, right, forward, backward

STEP 4: Place all the above Blocks in a loop that runs forever

Challenge 2 Solution



Note that if the action in the switch block is very quick (like adding to a variable or displaying a sensor value), the above loop and selected action will run multiple times

CREDITS

This tutorial was created by Sanjay Seshan and Arvind Seshan

More lessons are available at www.ev3lessons.com

INTERMEDIATE PROGRAMMING LESSON

CALIBRATING COLOR SENSORS

By Sanjay and Arvind Seshan

Lesson Objectives

1) Learn why you need to calibrate your color sensors
2) Learn what calibration is
3) Learn how to calibrate your color sensors

Why Calibrate?

When you use your EV3 Color Sensor in Light Sensor Mode (e.g., reflected light mode), you should calibrate it (not for Color Mode)

Calibration means "teaching" the sensor what is "Black" and what is "White"

- This makes White read as 100 and Black read as 0

Run your Calibrate Program whenever light conditions change

If you have 2 Color Sensors, the same calibration will apply to BOTH sensors. You don't have to make a different calibration program for each color sensor. Make it using 1 sensor on one of the ports and the values will apply to both.

- If you have sensors that are very different from each other, you will need to write your own custom calibration.

Steps/Pseudocode for Calibration

Challenge: Write a program that will calibrate your EV3 Color Sensors for black and white.

Pseudocode:

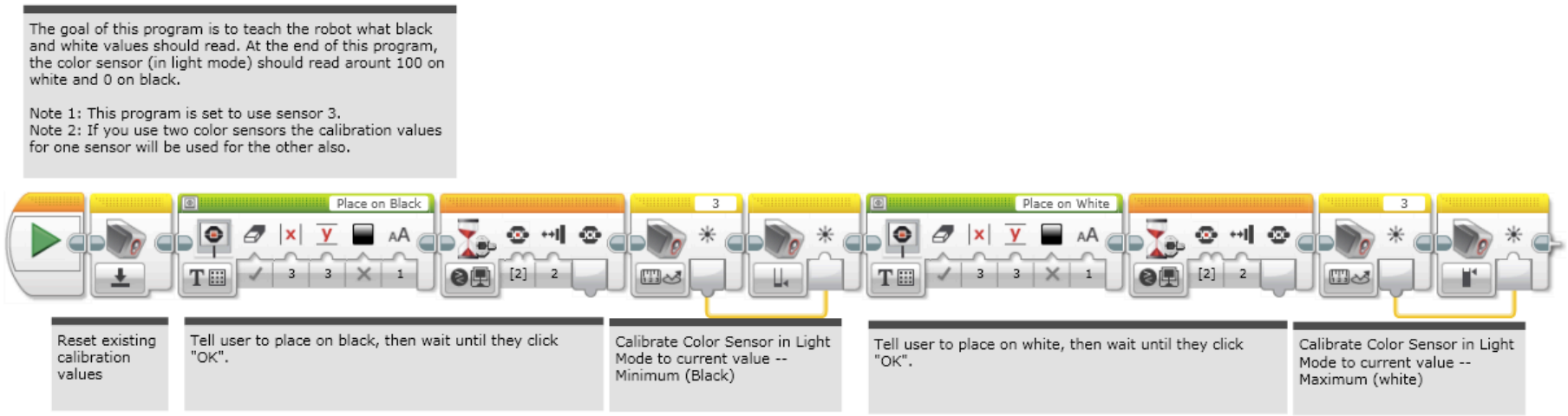
Reset the existing calibration values

Display that the user should place the robot on “black” and press ok

Read the Color Sensor Block in Light mode and save it to the Color Sensor Block in Calibrate mode.

Repeat above steps for calibrating “white”.

Calibrate Program Solution



- When you run the above Calibrate Program, you will be asked to place the robot on a BLACK section of the mat and then hit center EV3 button.
- Then you will be asked to place the robot on WHITE and hit center EV3 button.

Discussion Guide

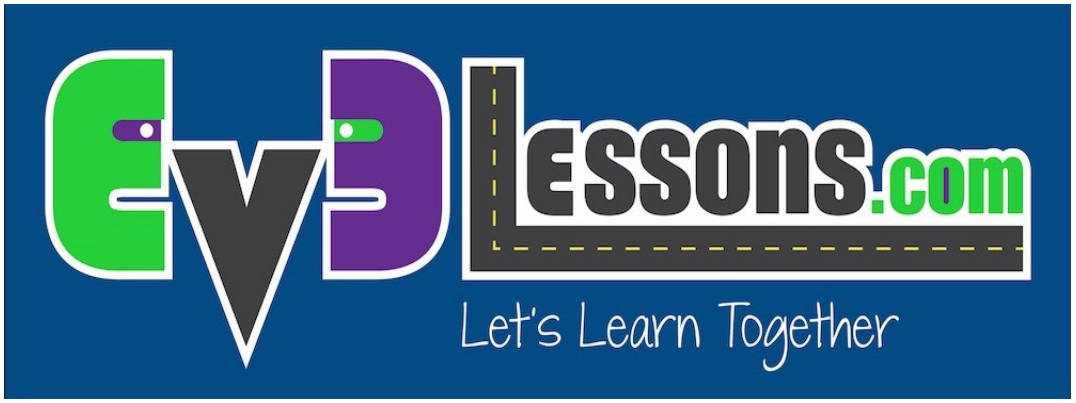
1. When do you need to calibrate your color sensor?
   - When it is used in reflected light mode
2. If I have two color sensors, do I need to calibrate each one?
   - The calibration applies to both (or all) the color sensors you have connected to your brick
3. What are you doing when you calibrate?
   - You are teaching the sensors what “black” and “white” mean
4. Should you calibrate for other colors (e.g. green) if you want to follow a green line?
   - No, you always calibrate for black and white.

Credits

This tutorial was created by Sanjay Seshan and Arvind Seshan

More lessons at www.ev3lessons.com

INTERMEDIATE PROGRAMMING LESSON



COLOR LINE FOLLOWER MY BLOCK WITH INPUTS: MOVE FOR DISTANCE

By Sanjay and Arvind Seshan

Lesson Objectives

1. Learn how to write a line follower that takes multiple inputs
2. Learn how to write a line follower that stops after a certain number of degrees
3. Practice making a useful My Block

Prerequisites: My Blocks with Inputs & Outputs, Data wires, Loops, Switches.

The code uses Blue Comment Blocks. Make sure you are running the most recent version of the EV3 Software. EV3Lessons has Quick Guides to help you.

My Block Line Follower with Inputs

- Making a My Block out of your line follower reduces the length of your code and makes it reusable
- Learning to write a line follower that takes multiple inputs (power, degrees and color) can be very useful
  - Every time you want a line follower that goes a different distance, you just need to change the input!

Tips to Succeed

You will need to know how to make a Simple Color Line Follower program and how to make a My Block with inputs

Since you will use your EV3 Color Sensor in Color Mode, you will not have to Calibrate your color sensor for this lesson

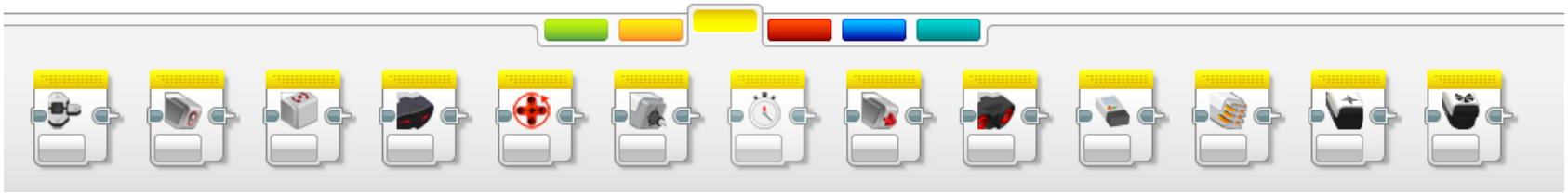
Check which ports you have your color sensor connected to and adjust the code as needed

You may have to adjust the speed or direction to work for your robot. Make sure that the the color sensor is in front of the wheels in the direction of travel.

Make sure you place the robot on the side of the line that you are following. The most common mistake is placing the robot on the wrong side of the line to begin with.

New Block

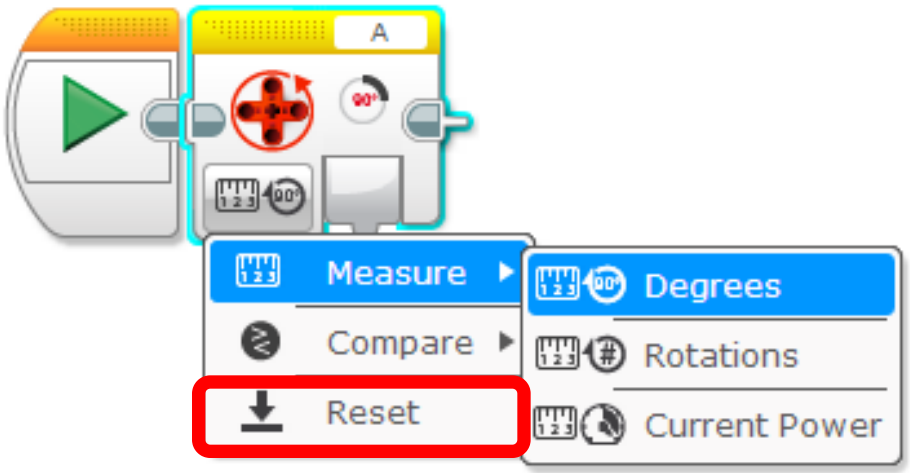
In this lesson, you will use the Sensor Block from the yellow tab for the first time.



We will use the Motor Rotation block. This is the rotation sensor.

The block has many useful modes.

In this lesson, we learn to use it in reset mode so that the value in the sensor will be set to 0.



Color Follower for Distance

STEP 1: Create a simple color line follower program

STEP 2:

A. Include a “reset the rotation” sensor block to delete any prior readings

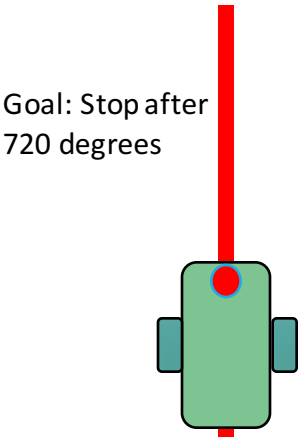
B. Exit the line follower loop when the robot has moved certain degrees

STEP 3:

A. Create a My Block with the code in Step 2 with inputs for degrees, power and color.

B. Wire the inputs in the My Block

Challenge: Write a line follower My Block that follows a colored line and stops after moving a certain number of degrees. The line follower should take three inputs (degrees, power and color to follow).

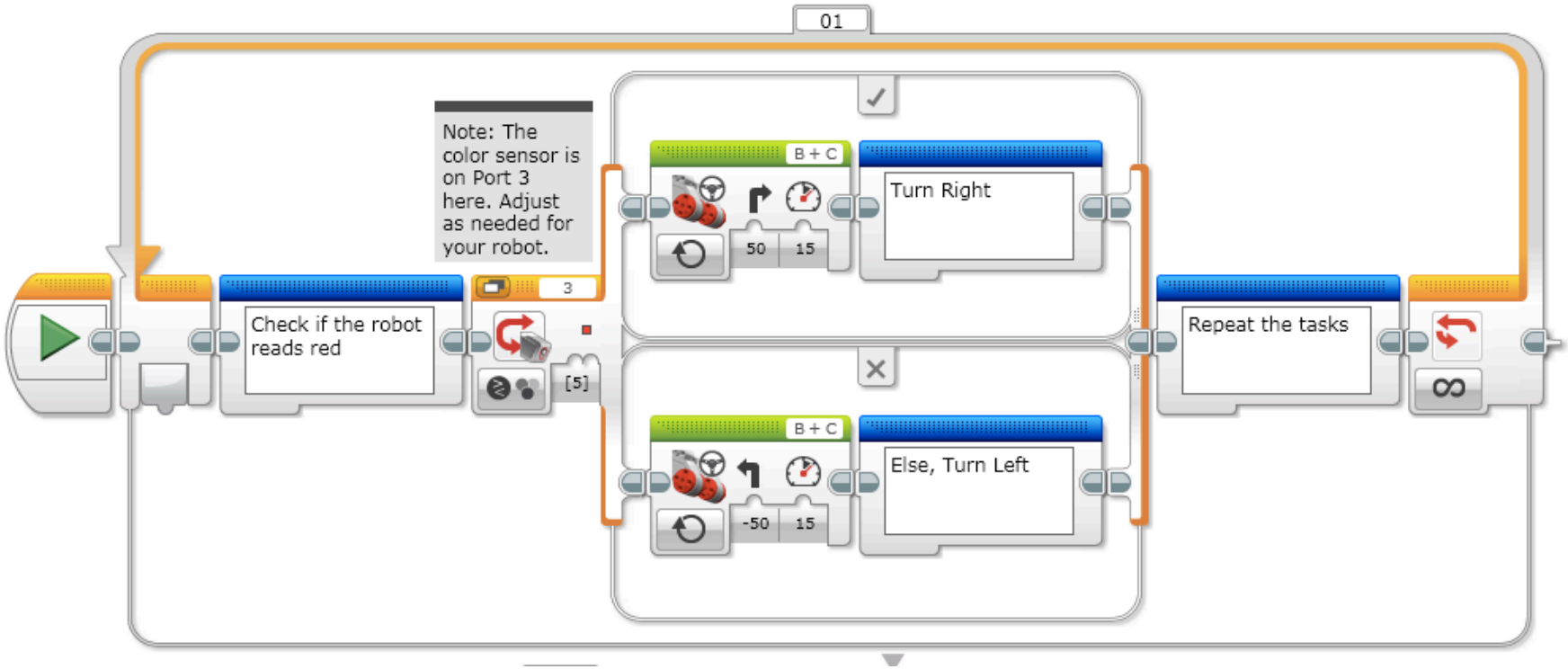


Step 1: Simple Color Line Follower

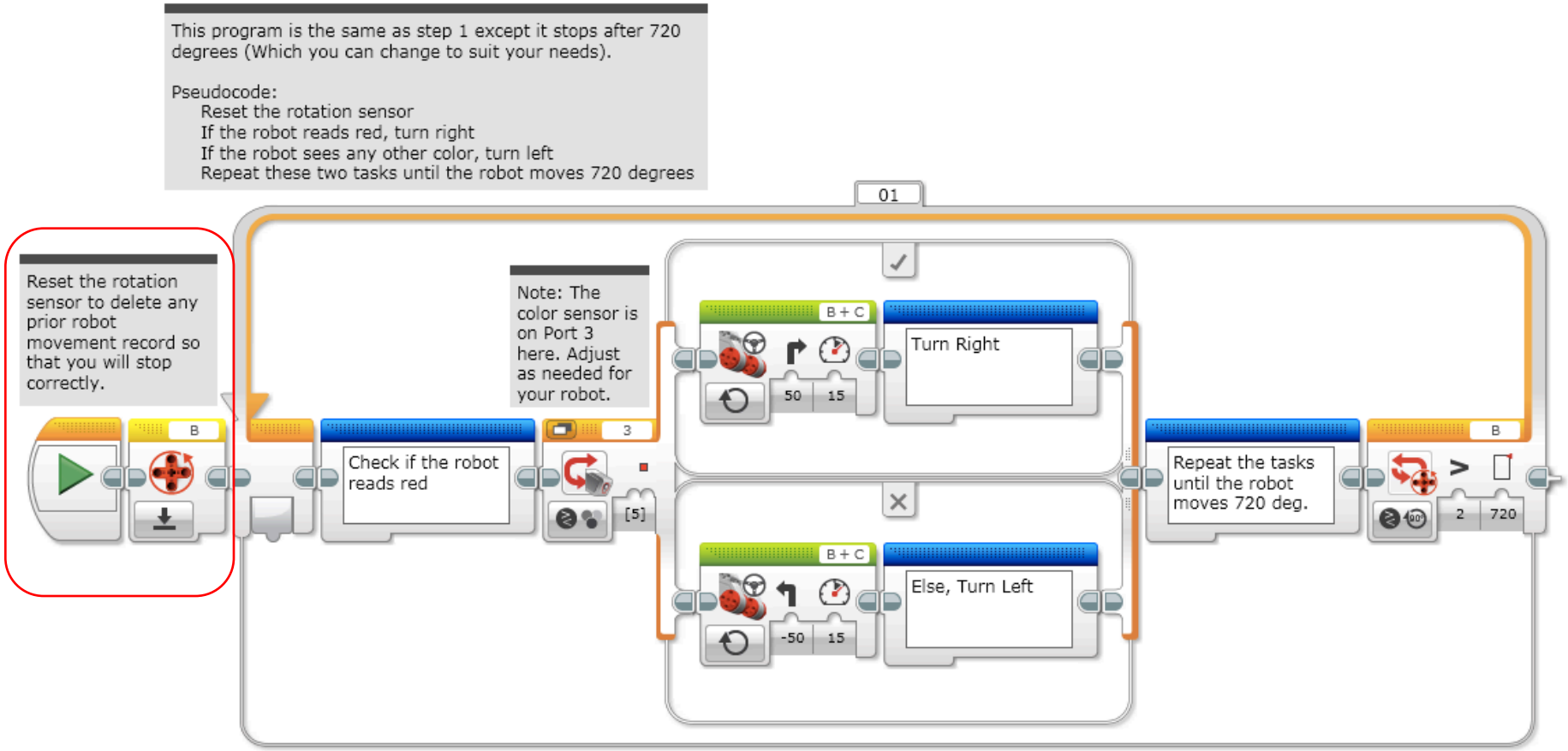
Goal: To create a Line Follower with Color as the input.

Step 1: Create a simple color line follower that follows the right side of the line.

Pseudocode:
If the robot reads red, turn right
If the robot sees any other color, turn left
Repeat these two tasks



Step 2: Add Reset & Loop Exit

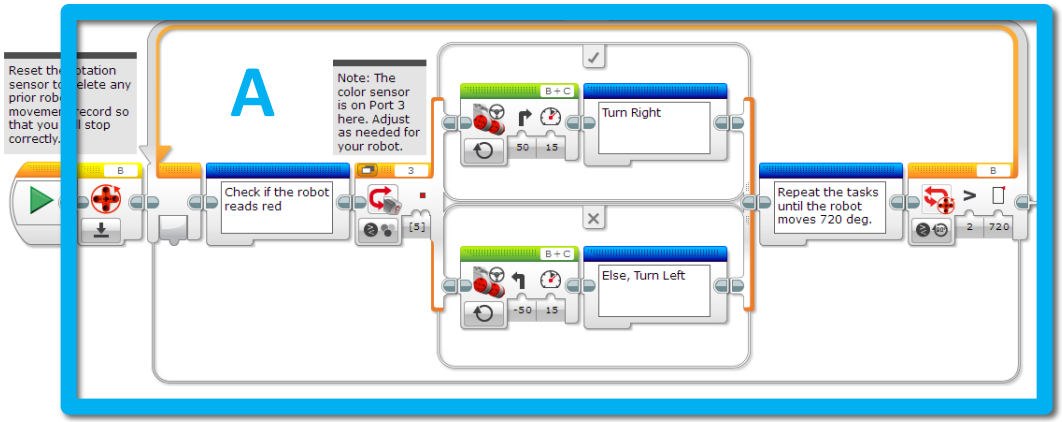


Step 3a: Create a My Block

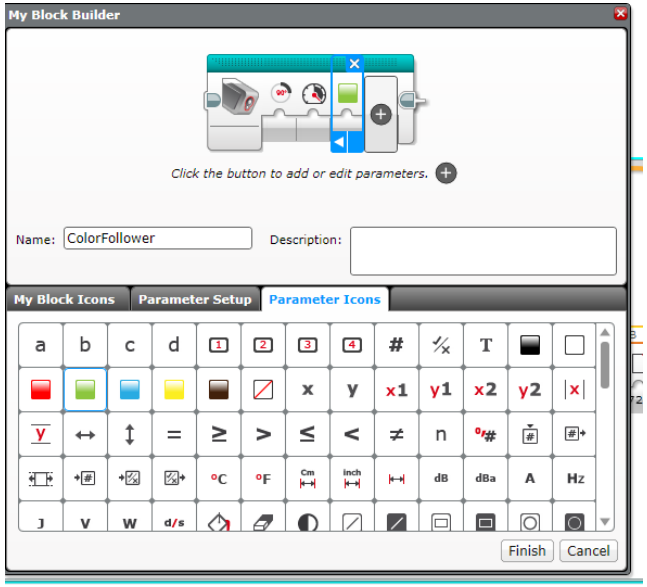
A. Highlight all the blocks then go to My Block Builder

B. Add 3 inputs: one for power and one for color, and one for degrees

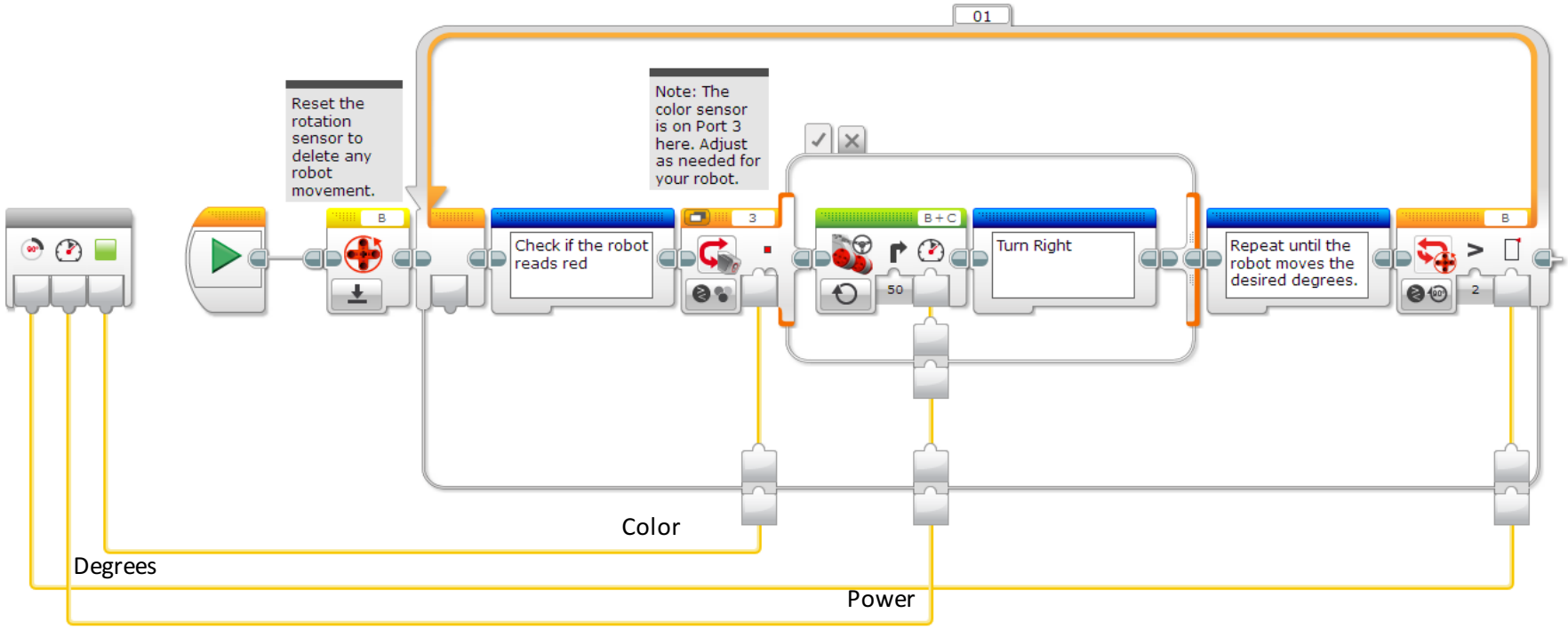
- Refer to the My Blocks with Inputs & Outputs lesson if you need help setting up the My Block



B



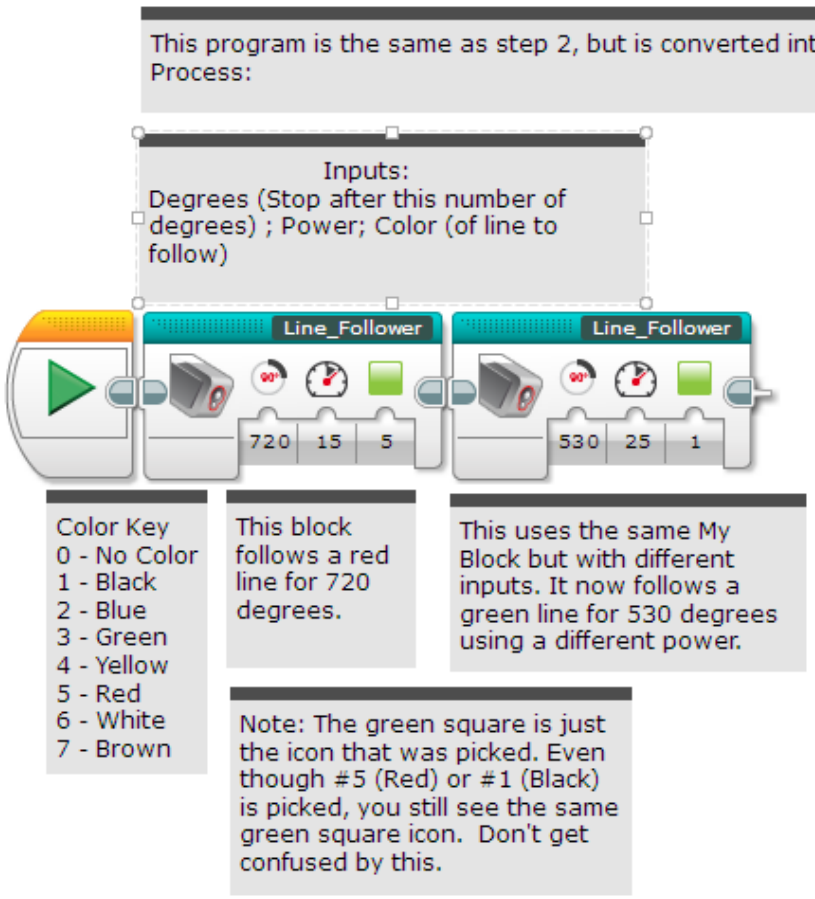
Step 3B: Wire the My Block



C

- The degrees input goes into loop exit condition
- The power input goes into power input on the steering block
- The color input goes into color input for the switch

STEP 3C: Using the My Block



Now the My Block appears in the turquoise tab and the same My Block can be used again and again with new inputs (see left)

- The first block alone solves the challenge of line following for 720 degrees.
- The second block in this code is to show that the same block can be used with different inputs to follow a different line for a different distance.
- If you want to learn smoother line followers, proceed to the proportional control lesson in Advanced.

Credits

This tutorial was created by Sanjay Seshan and Arvind Seshan

More lessons are available at www.ev3lessons.com

INTERMEDIATE PROGRAMMING LESSON

COLOR LINE FOLLOWER MY BLOCK WITH INPUTS: MOVE UNTIL BLACK

By Sanjay and Arvind Seshan

Lesson Objectives

1. Learn how to write a line follower that takes multiple inputs
2. Learn how to write a line follower that stops when it sees a another line
3. Practice making useful My Blocks

Prerequisites: My Blocks with Inputs & Outputs, Data Wires

The code uses Blue Comment Blocks. Make sure you are running the most recent version of the EV3 Software. EV3Lessons has Quick Guides to help you.

Tips to Succeed

1. You will need to know how to make a Simple Color Line Follower program and how to make a My Block with inputs
2. Since you will use your EV3 Color Sensor in Color Mode, you will not have to Calibrate your color sensor for this lesson
3. Check which ports you have your color sensor connected to the EV3 and adjust the code as needed
4. You may have to adjust the speed or direction to work for your robot. Make sure that the the color sensor is in front of the wheels in the direction of travel.
5. Make sure you place the robot on the side of the line that you are following. The most common mistake is placing the robot on the wrong side of the line to begin with.

Color Follower Until Color

Challenge: Create a line follower My Block that stops when it sees black

STEP 1:

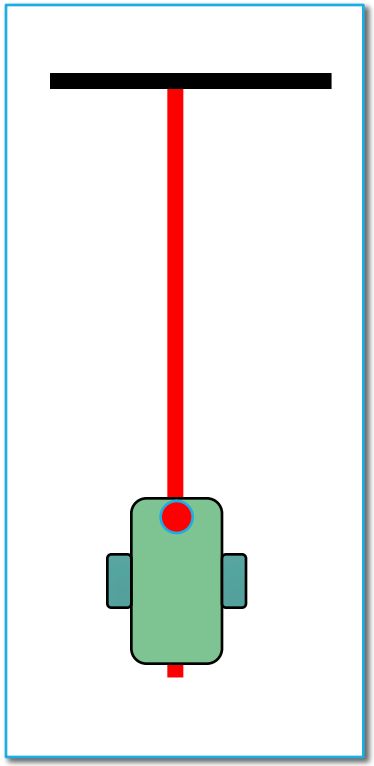
- Create a simple line follower

STEP 2:

- Change the loop exit condition to "until black"

STEP 3:

- A. Make a My Block with 3 inputs: Power, Color to line follow on, and Color to stop at
- B. Wire the My Block

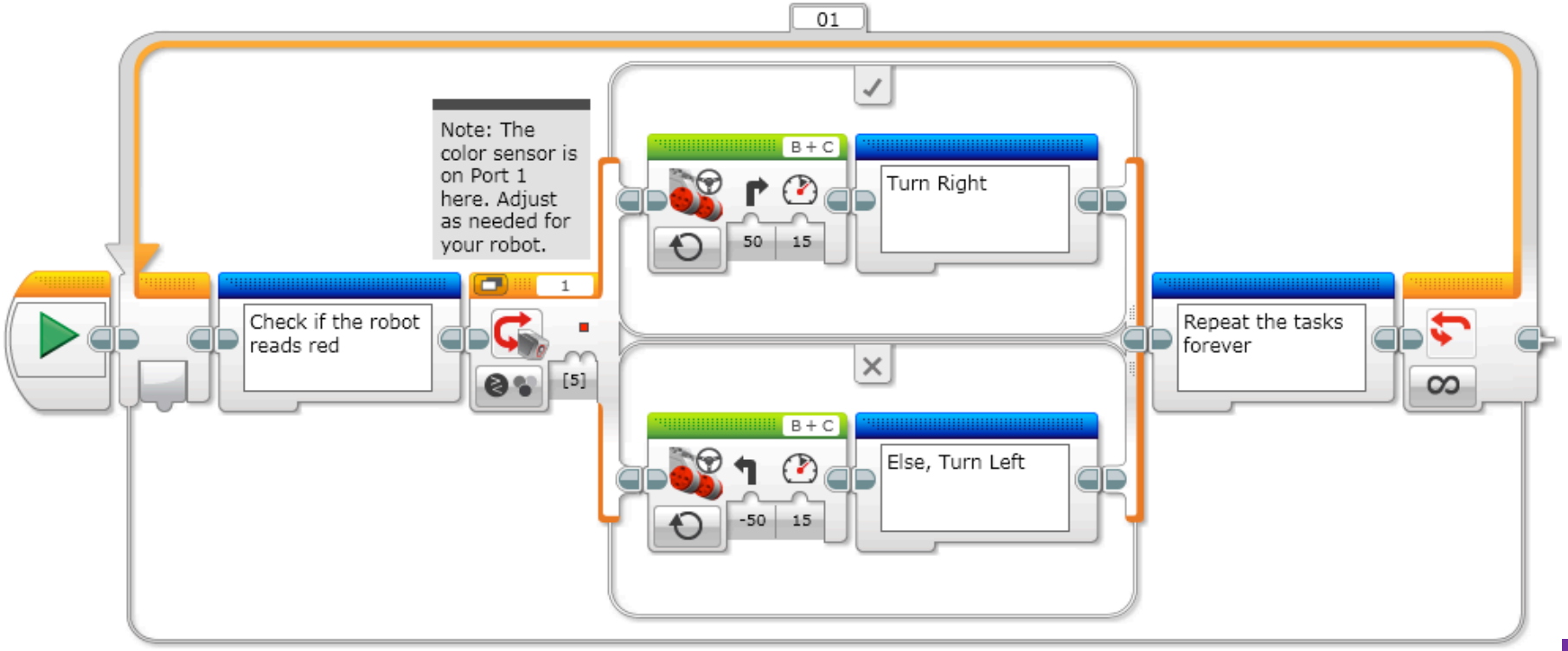


Step 1: Simple Line Follower

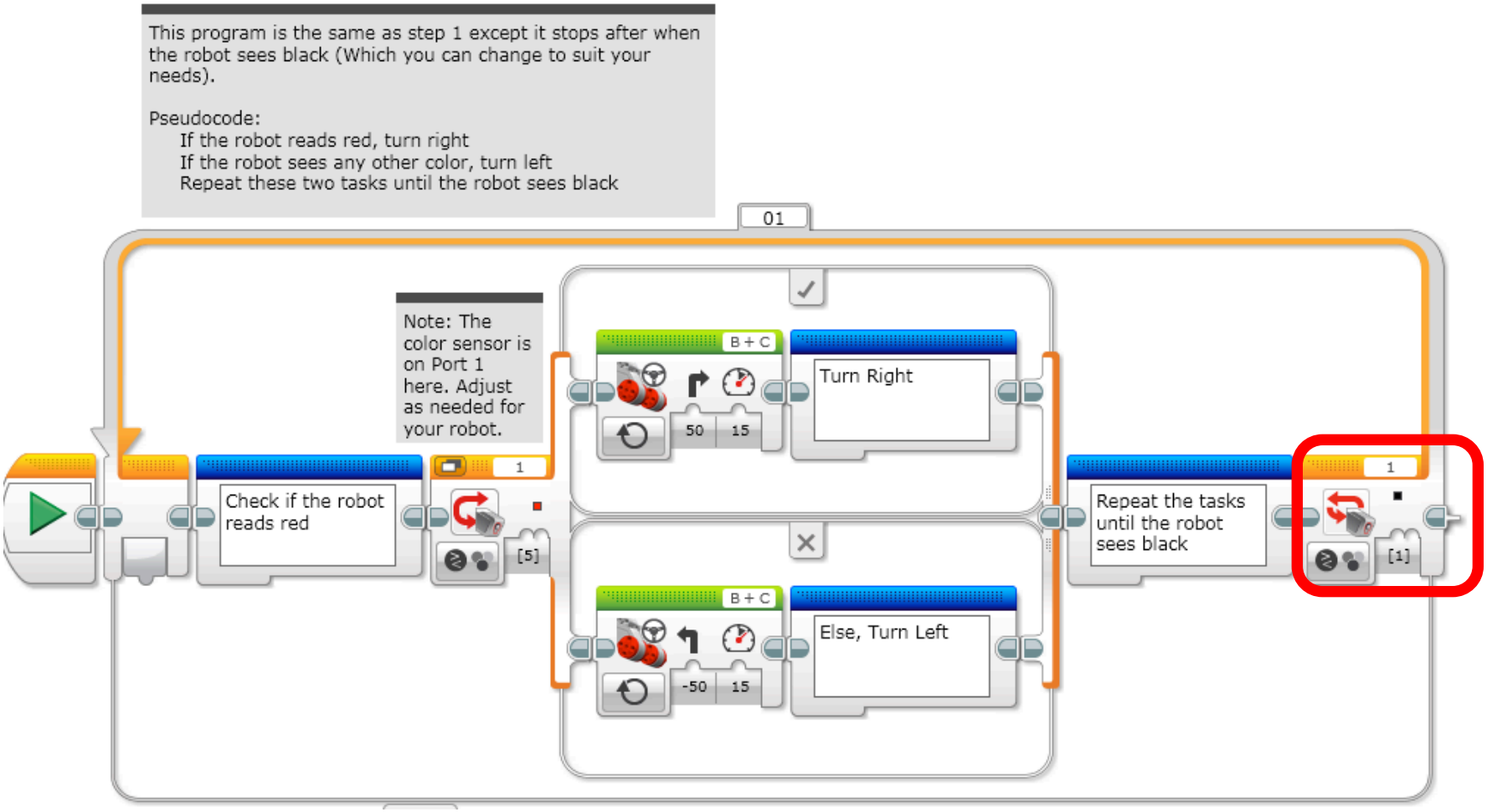
Final Goal: To create a Line Follower with Color as the input and stops on a black line.

Step 1: Create a simple color line follower that follows the right side of the line.

Pseudocode:
If the robot reads red, turn right
If the robot sees any other color, turn left
Repeat these two tasks



Step 2: Loop Exit Condition

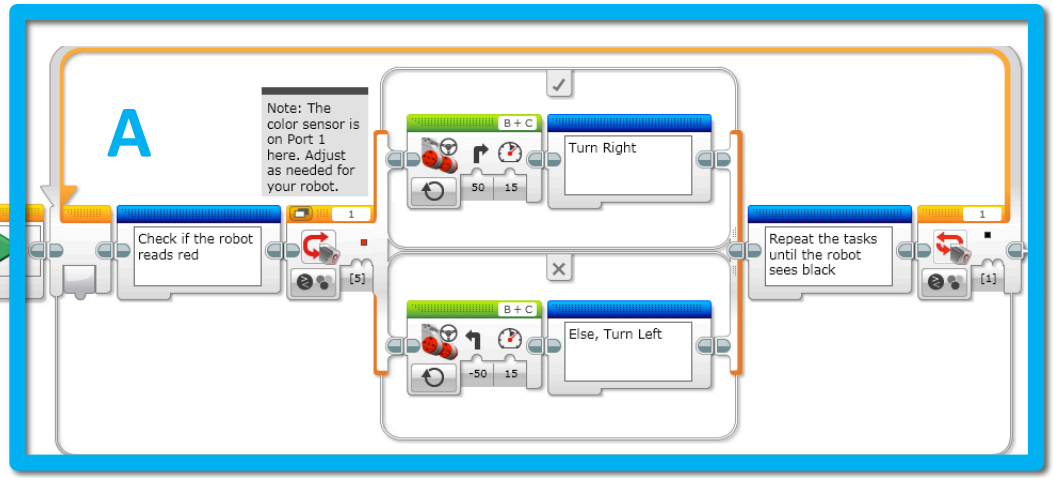


Step 3A: Create a My Block

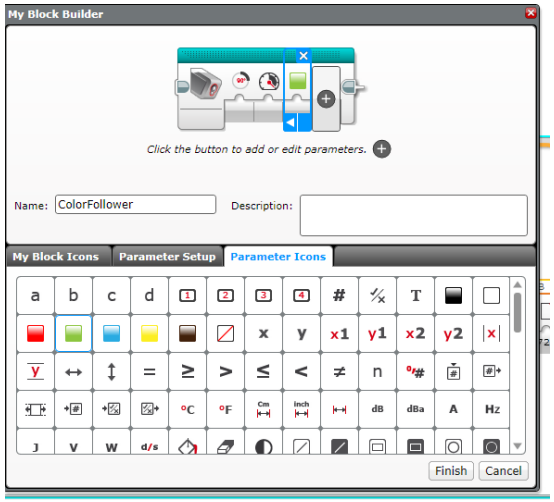
A. Highlight all the blocks then go to My Block Builder

B. Add 3 inputs: one for power and one for color, and one for degrees

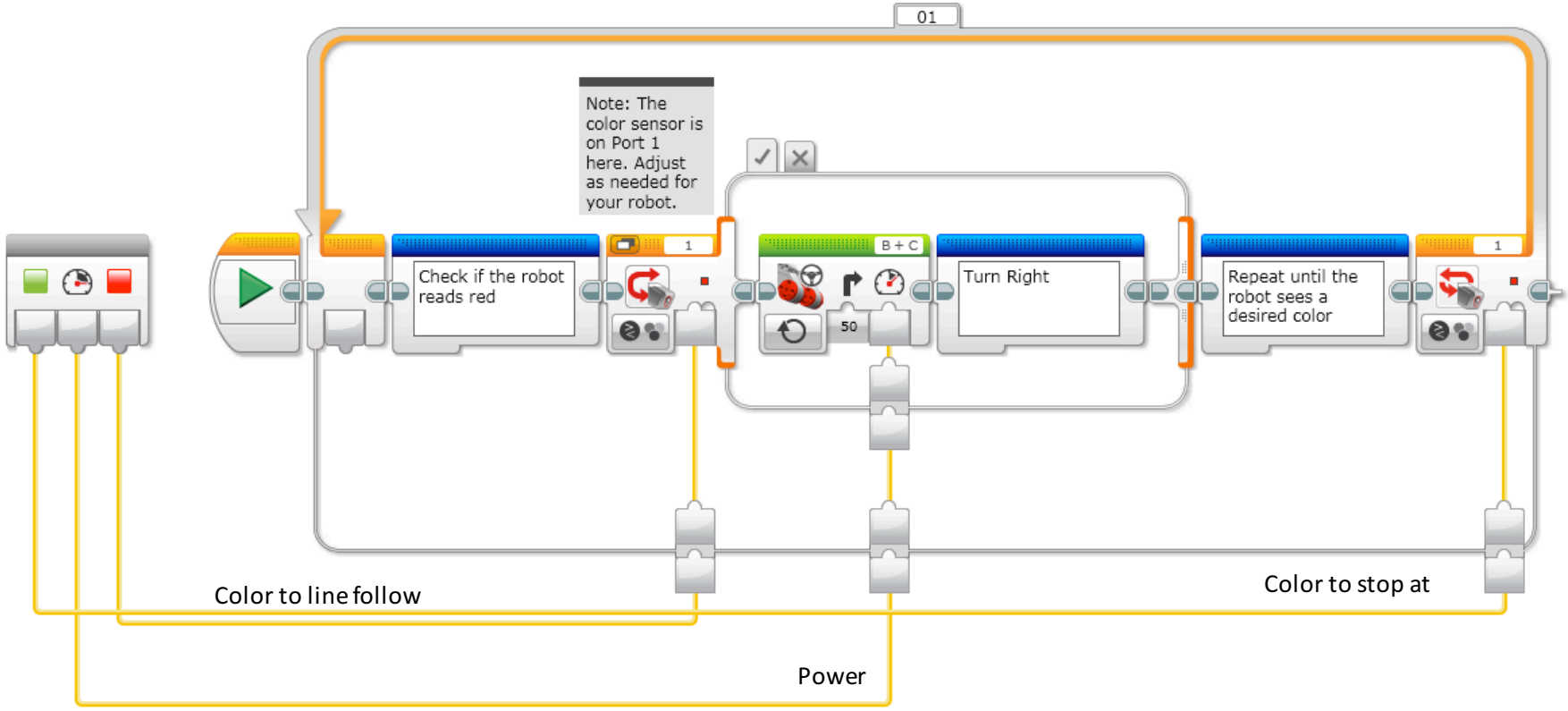
- Refer to the My Blocks with Inputs & Outputs lesson if you need help setting up the My Block



B



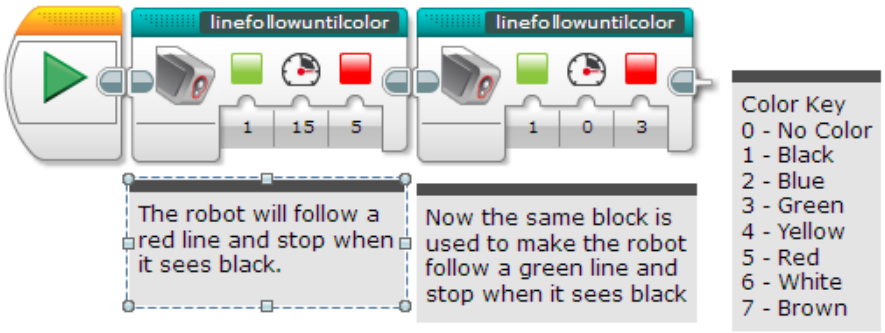
Step 3B: Wire the My Block



C

- The color to stop at goes into loop exit condition
- The power input goes into power input on the steering block
- The color input goes into color input for the switch

Step 3B: The My Block



- Now the My Block appears in the turquoise tab and the same My Block can be used again and again with new inputs (see left)
- The first block solves the challenge and follows a red line until it sees the robot sees black
- The second block in this code is to show that the same block can be used with different inputs
- If you want to learn smoother line followers, proceed to the proportional control lesson in Advanced.

Next Steps

- We use a simple line follower in this lesson. You can combine these techniques with any line follower.
- Learn how to create a proportional line follower for light or a smooth line follower for color → check out our Advanced: Proportional Line Follower lesson.

Credits

This tutorial was created by Sanjay Seshan and Arvind Seshan

More lessons are available at www.ev3lessons.com

INTERMEDIATE PROGRAMMING LESSON

DATA WIRES

By Sanjay and Arvind Seshan

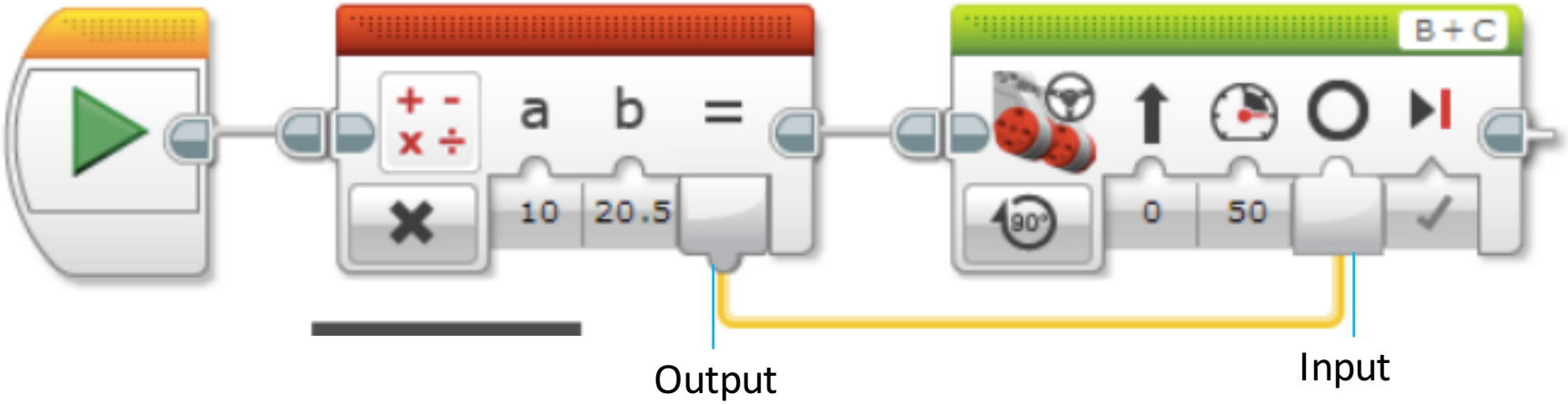
Lesson Objectives

Learn what Data Wires are and how to use them

Prerequisites: Display Block, Sensor Block, Brick Buttons

Data Wires

A Data Wire allows you to take an output from one programming block and input it into another.



Data Wire Types

Data Type	Input	Output	Output Data Wire
Logic		True or False 	
Numeric		Number 	
Text		Text 	
Numeric Array			
Logic Array			

Images from EV3 Help

Automatic Data Wire Conversions

From Data Type	To Data Type	Output/Result
Logic	Numeric	False = 0, True = 1
Logic	Text	False = “0”, True = “1”
Logic	Logic Array	Array with one element
Logic	Numeric Array	Array with one element (0 or 1)
Numeric	Text	Text that represents a number
Numeric	Numeric Array	Array with one element
Logic Array	Numeric Array	Same size array with all elements equal to 0 or 1

These conversions are automatically performed in the programming blocks. For example, you are allowed to connect a numeric value (like what color a sensor sees) to a text value (on a display block).

Content from EV3 Help

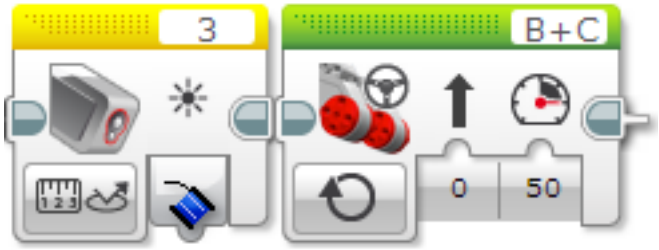
How to Create a Data Wire

The block with the output must be placed before the block with the input

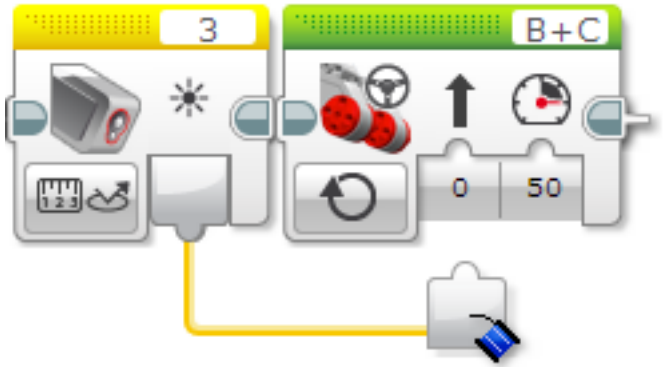
The input and the output must be the same data type or one that can be automatically converted (see slides 4 and 5)

1. Click on the output on the block
2. Hold and drag the wire.
3. Move the icon into the correct input and then let go of the mouse

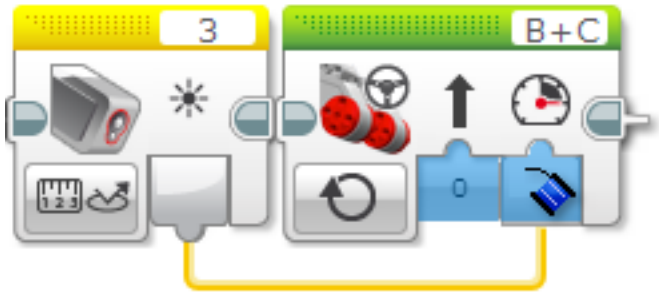
1



2



3

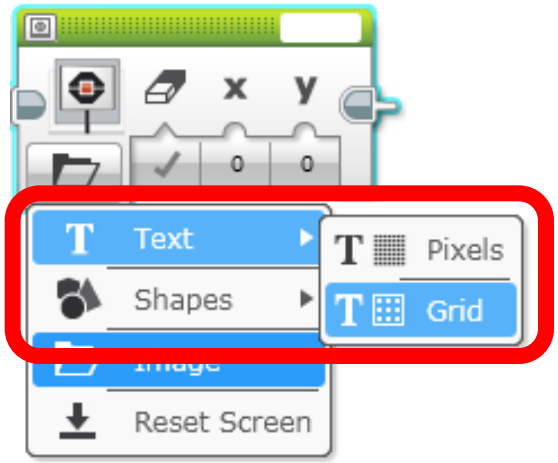


Images from EV3 Help

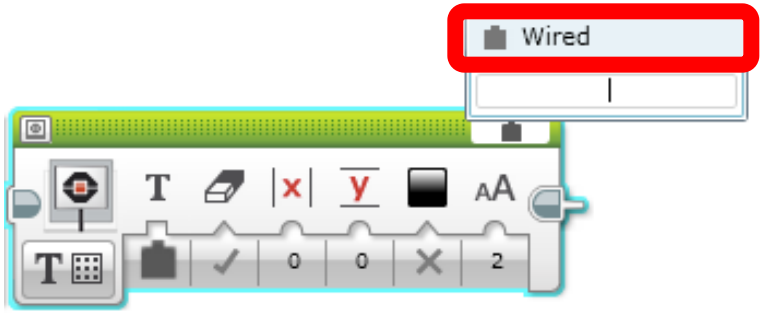
Sidebar: Display Block - Wired Mode

The Display Block can be used in wired mode to display data from another block to the screen.

For the challenge, you will need to display a number on the screen. Pick Text Mode →Grid from the bottom left corner of the block.



To pick Wired Mode, click on the top right corner of the Display Block and pick wired



Data Wires Challenge

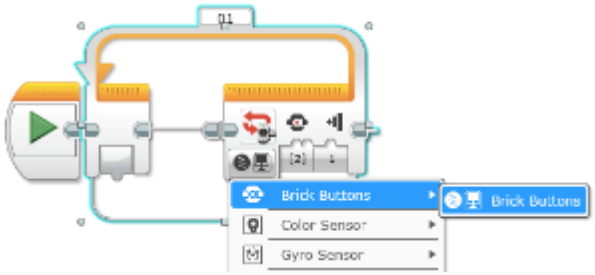
CHALLENGE: Make your robot drive forward slowly over different colors. Have the robot display the color the color sensor sees as it moves. Stop when you hit a button on the brick.

STEP 1: Turn the motors on in a Steering Block and drive slowly forward

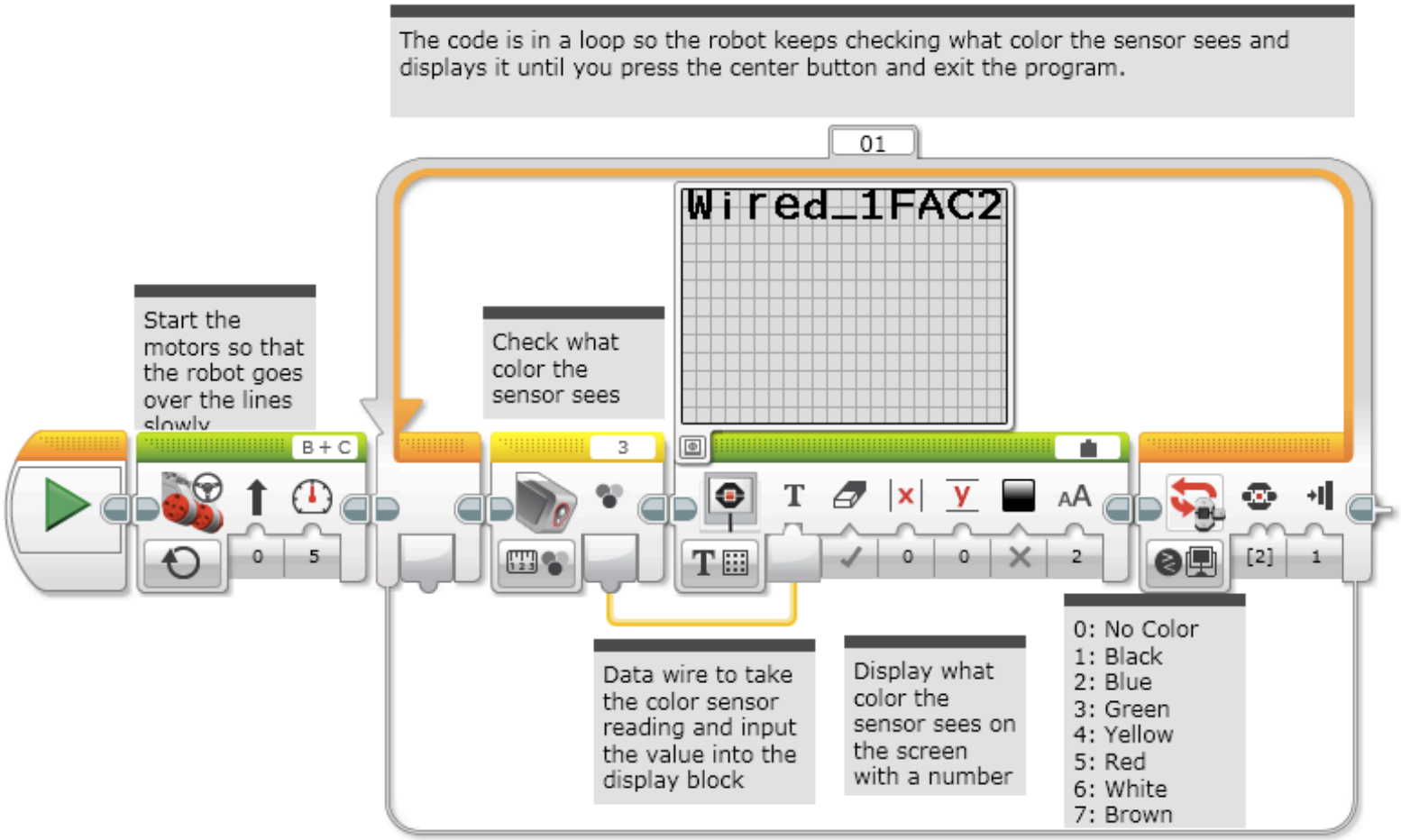
STEP 2:

- Inside a Loop, add a Color Sensor block.
- Add a Display Block in Wired, Text Grid Modes.
- Wire the Sensor Block's output into the Display Block's text input (first input)

STEP 3: Exit the loop when a brick button is pressed

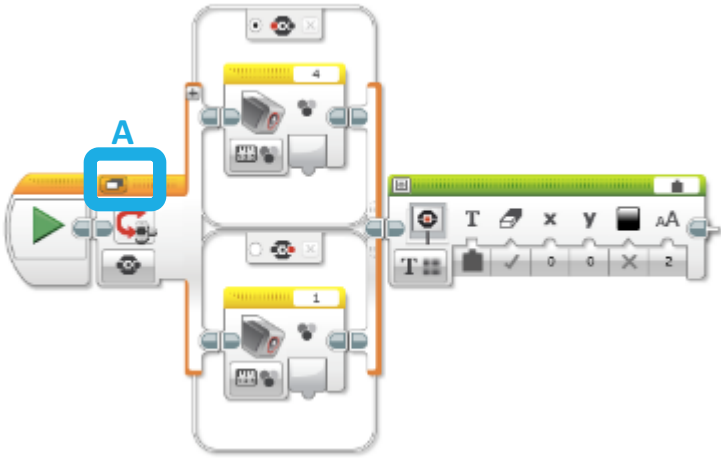


Challenge Solution

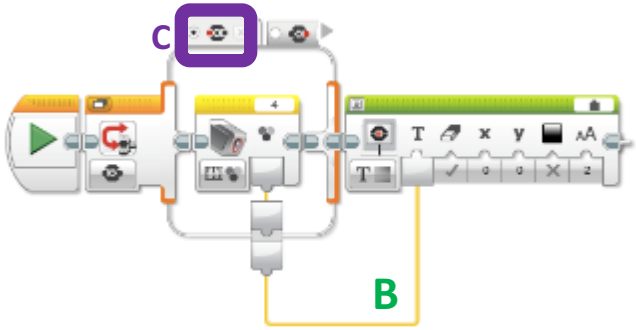


More Complex Wiring: Switches

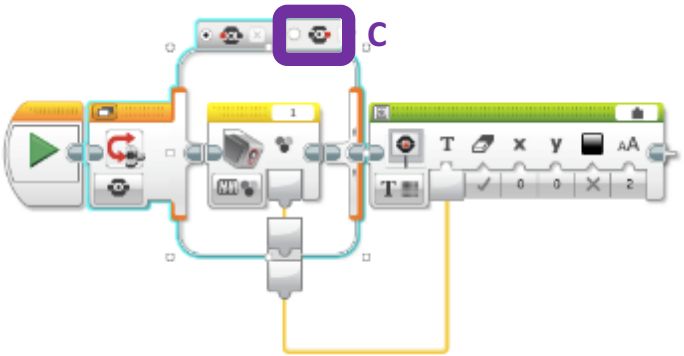
A. If you want to drag data wires out of switches, you will need to change the switch to tabbed view



B. Once you switch to tabbed view, you can drag data wires out

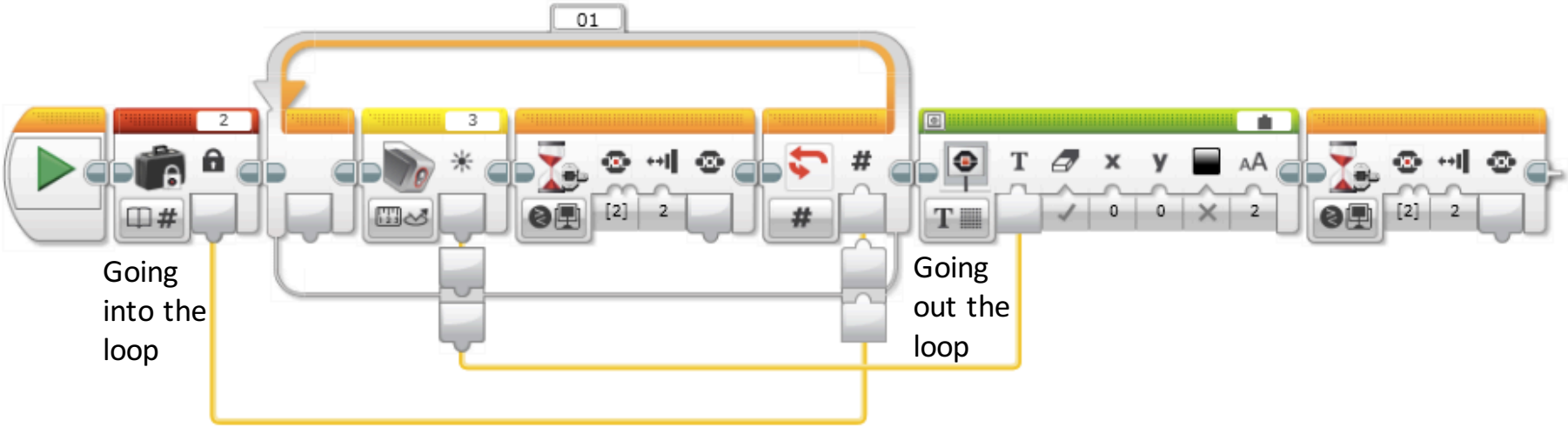


C. Different options in the switch can connect to the same wire



More Complex Wiring: Loops

You can connect wires both into and out of a loop like in the example below

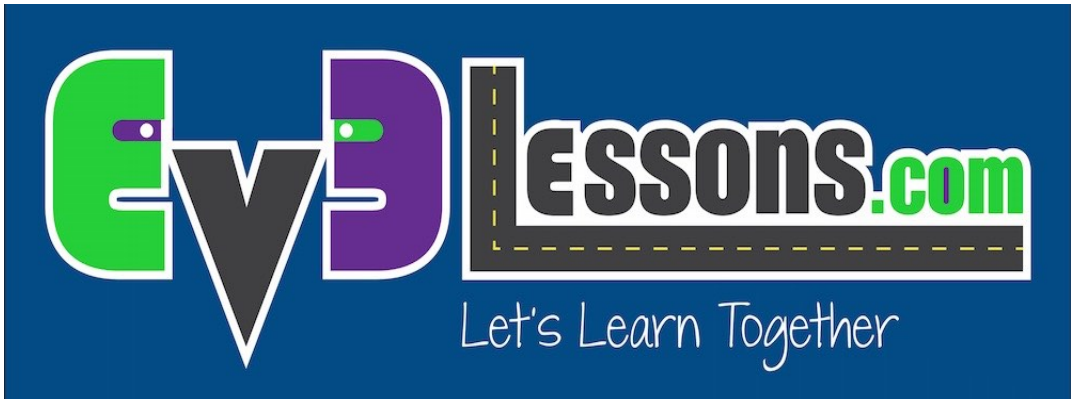


- Note that the data coming out of the loop through the wire will only be the last pass through the loop.
- In the example above, the color sensor is read twice in the loop. However, the data wire will only have the second (and last) reading and that second reading will be displayed.

Credits

- This tutorial was written by Sanjay and Arvind Seshan
- More lessons at www.ev3lessons.com

INTERMEDIATE PROGRAMMING LESSON



DEBUGGING TECHNIQUES

By Sanjay and Arvind Seshan

Lesson Objectives

1) Learn the importance of debugging
2) Learn some techniques for debugging your code

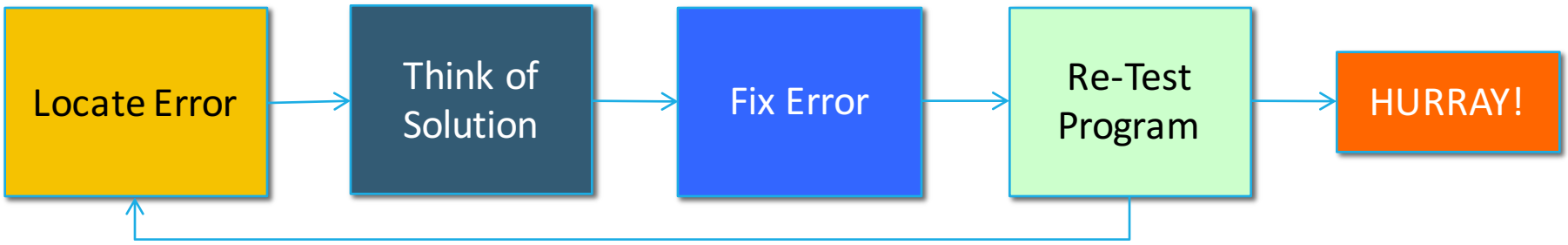
Why Debug?

Debugging is a useful strategy to figure out where in your program something is going wrong or what went wrong

Once your code starts to become long or complicated (e.g. using sensors), it can become hard to figure out where in the program you are

The following slides show you some ways of knowing where you are in your program or knowing what values your sensors see

You will see that these techniques can be VERY USEFUL to any programmer.



Different Techniques

Play Selected vs. Button Press

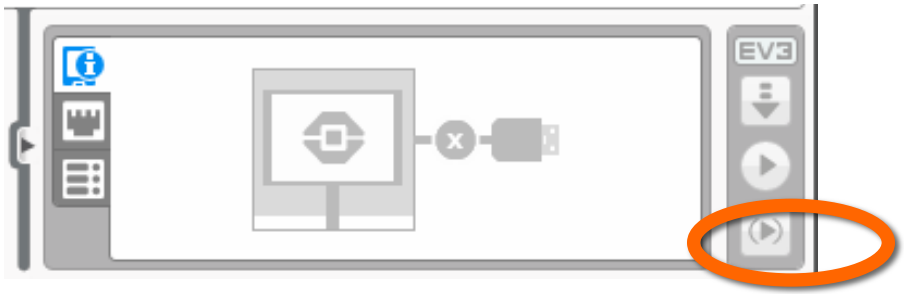
- Very similar techniques
- Lets you try out smaller portions of code
- Play Selected requires bluetooth
- Button Press requires some care so you don't jostle the robot when pressing the button

Light, Sound and Display

- Very similar techniques
- Light and Sound are used in the same way
- Teams enjoy the sound more and it is easier to identify sometimes
- Display Block comes in handy for knowing what block is played if your robot gets stuck and if you want to see the sensor values

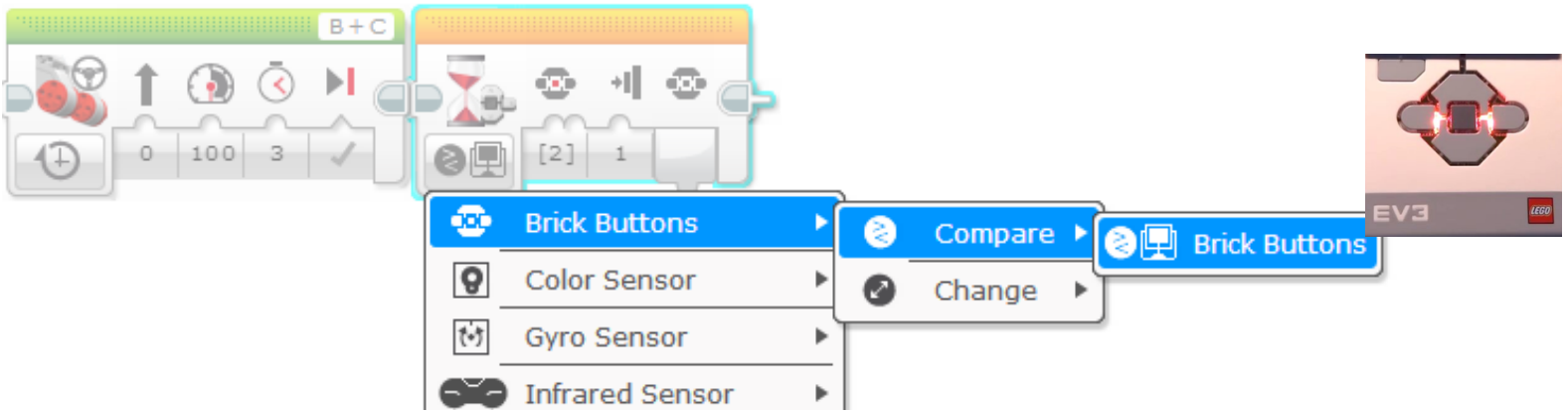
Play Selected

- Play selected is useful for running small parts of the program
- Use when you don't want to wait for your robot to complete other parts of the program before getting to the part you want to see
- If you don't have bluetooth built in the computer, we recommend that you purchase a bluetooth dongle (US $10-15) because it makes this type of debugging easier
- To use, highlight the parts of the program you want to run and pick the play button with the parentheses (>)

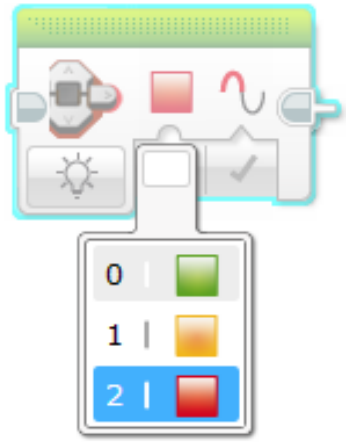


Wait For Button Press

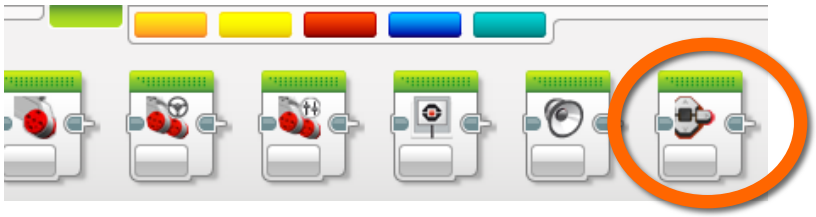
- To place a Wait for Button Press block in your program, place a wait block into your program
- Go under brick buttons > compare > brick buttons, then choose which button needs to be pressed to continue the program
- Place these wait for button presses every block or two close to where the robot is not working correctly
- This can help you pinpoint which block is causing the robot to fail
- The robot will stop and “wait for you to press the button”



Visual Alerts: Brick Status Light



- Brick status light blocks can be used for warnings

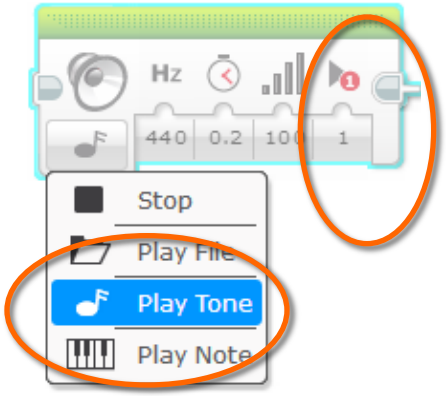


Brick Status Light block

- Place these blocks at critical steps in your program
- You will then be able to visualize what block is playing and figure out where the error might be

Sound Alerts: Sound Block

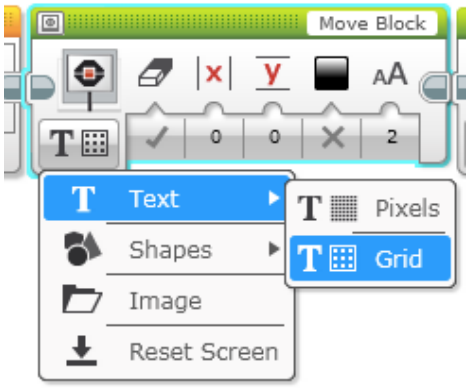
- You can insert different sounds at intervals (about every 5 blocks or so, and then run the program again while listening for beeps.
- Once you pick Play Tone, select Play Type and pick “play once”
- These sounds can help you narrow down where in the program something is going wrong.

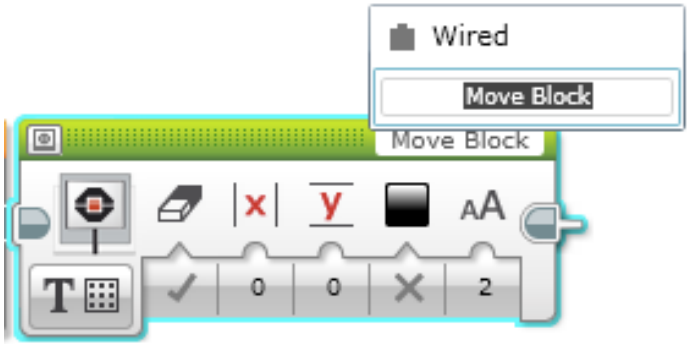


Sound block

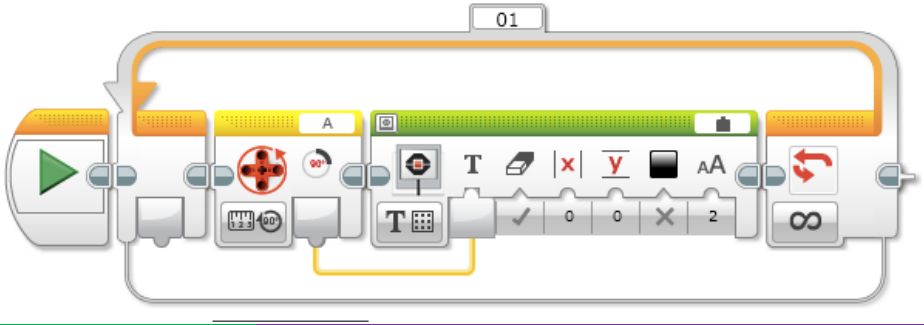
Print to Screen: Display Block

- Showing which block is playing on your robot
  - Helps identify what block the robot is stuck on





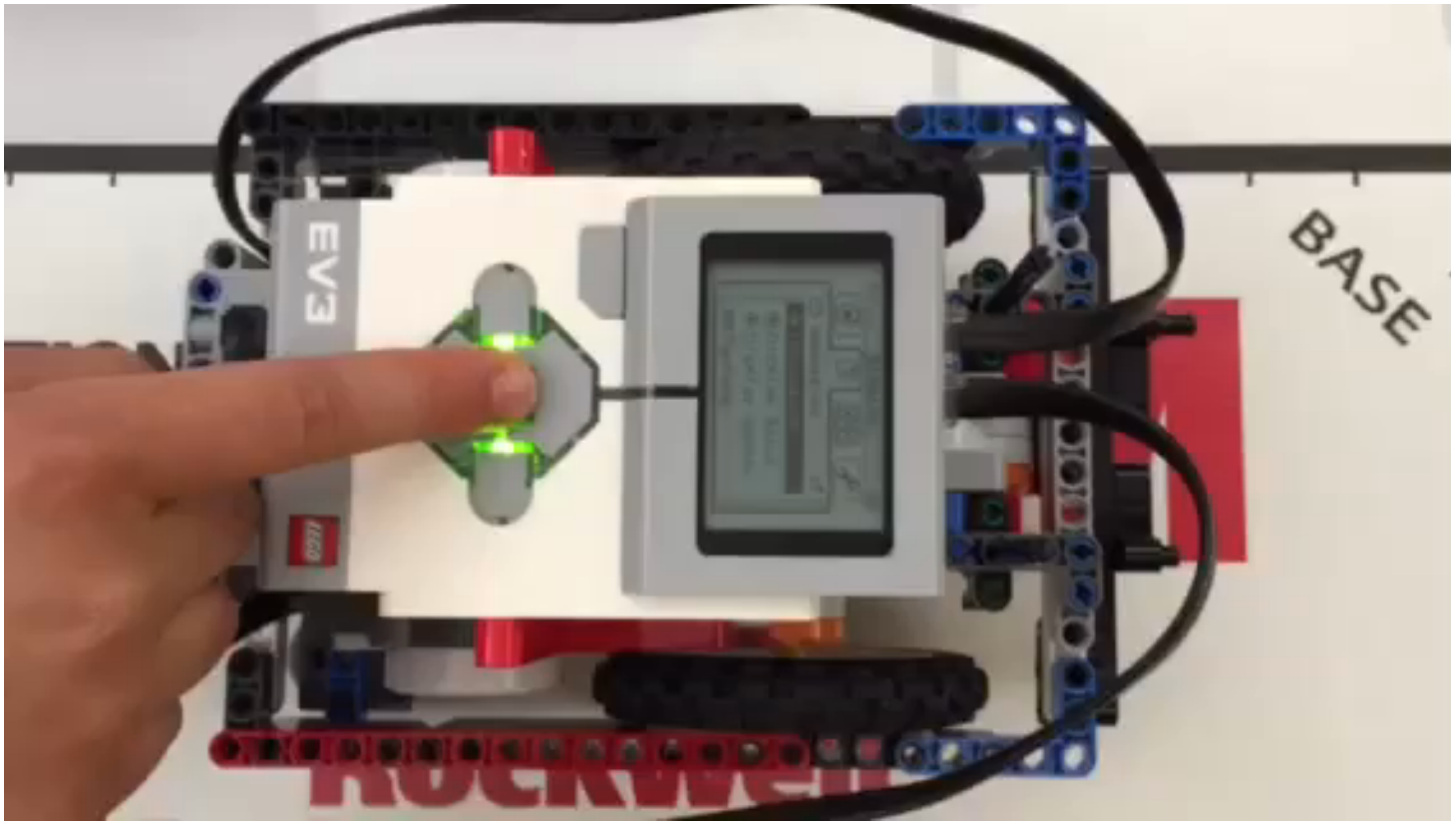
- Seeing the sensor readings – to see what the robot sees!



Sample Video on Next Slide

- The video on the next slide shows some of the debugging techniques
  - Wait for button press
  - Sounds alerts
  - Brick lights
  - Sensor readings displayed on brick

Sample Video – Click to Play



Other Methods

- Recordings:
  - You can record your robot with a camera. Then watch the video and observe what went wrong

- Comments:
  - You can also use “comments” to help debug – we add comments to remember what older values were entered into a block. We watch the robot and then adjust these values

CREDITS

This tutorial was created by Sanjay Seshan and Arvind

Author's Email: team@droidsrobotics.org

ADVANCED EV3
PROGRAMMING LESSON

Using the Gyro Sensor and Dealing with Drift

By Sanjay and Arvind Seshan

Lesson Objectives

1. Learn what the Gyro Sensor does
2. Learn about 2 common problems with using the gyro sensor (drift and lag)
3. Learn what “drift” means
4. Learn how to correct for drift with a gyro “calibration” technique
5. Understand why it is important to consider multiple solutions to a problem such as gyro drift

Prerequisites: Data wires, Loops, Logic & Comparison Blocks

What is the Gyro Sensor?

- Gyro sensor detects rotational motion
- The sensor measures the rate of rotation in degrees per second (rate)
- It also keeps track of the total rotational angle and therefore lets you measure how far your robot has turned (angle)
- The accuracy of the sensor is ±3 degrees for 90 degree turn

Gyro Sensor Problems

- There are 2 common Gyro issues – drift and lag
  - Drift – readings keep changing even when the robot is still
  - Lag – readings are delayed
- In this lesson, we focus on the first problem: drift.
  - We will cover lag in the Gyro Turn lesson
- Solution to drift: gyro calibration
  - The source of the drift problem is that the gyro must "learn" what is still.
  - For a color sensor, you have to "teach" the robot what is black and white
  - For your gyro, you need to calibrate the sensor to understand what is "still"

Gyro Calibration to Solve Problem 1: Lag

- The gyro auto-calibrates when the robot is turned on or the gyro wire is connected. If the robot is moving during calibration, the gyro “learns” the wrong value for “still” – this causes drift!
- Unfortunately, there is no gyro calibration block. There a few ways to make the sensor recalibrate.

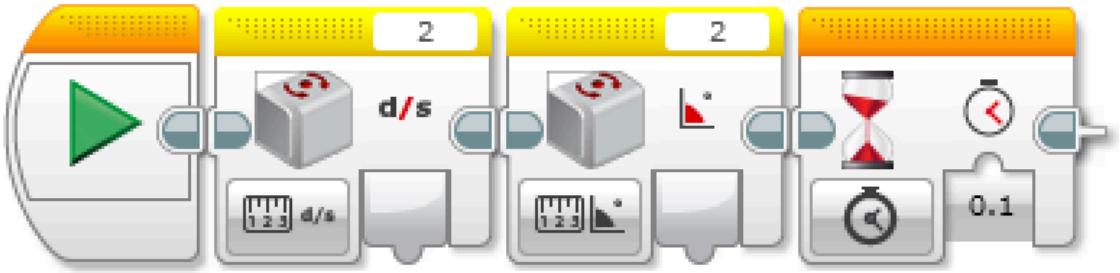
IMPORTANT NOTES

- The below are critical notes for using the gyro correctly!!!!!
- THE ROBOT MUST BE STILL WHEN YOU RUN ANY OF THESE CALIBRATION PROGRAMS!!!!
- JUST LIKE THE COLOR CALIBRATION, YOU SHOULDN'T RUN THIS EVERY TIME YOU NEED TO READ THE GYRO. YOU SHOULD CALIBRATE IN A SEPARATE PROGRAM JUST BEFORE YOU RUN YOUR PROGRAM OR ONCE AT THE BEGINNING OF YOUR PROGRAM.

Calibration: Strategy 1

The gyro recalibrates when it switches modes. So, a “rate” reading followed by an “angle” reading calibrates the gyro.

Second, add a wait block to give the sensor a bit of time to fully reset. Our measurements show that 0.1 seconds is sufficient.



By reading the rate of the gryo then the angle the gyro sensor gets recalibrated. Make sure to run these blocks when the robot is still.

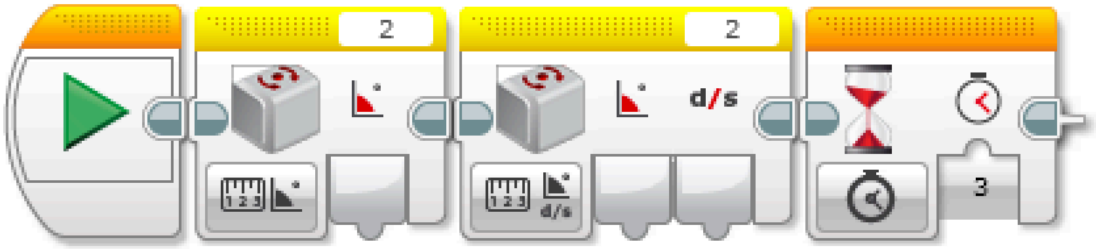
This is a wait block. We wait 0.1 seconds because it takes time for the gyro sensor to reset to zero.

Note that in the rest of your program, you should only use the “angle” modes of the gyro. Using the “rate” or “rate and angle” mode will cause the gyro to recalibrate.

Calibration: Strategy 2

This version of the calibration leaves the gyro in rate+angle mode. This is useful if you use the rate output.

The downside of this version is that it takes longer (about 3 seconds). Also, you cannot use gyro reset anymore!



By reading the angle of the gryo then the rate+angle the gyro sensor gets recalibrated. Make sure to run these blocks when the robot is still.

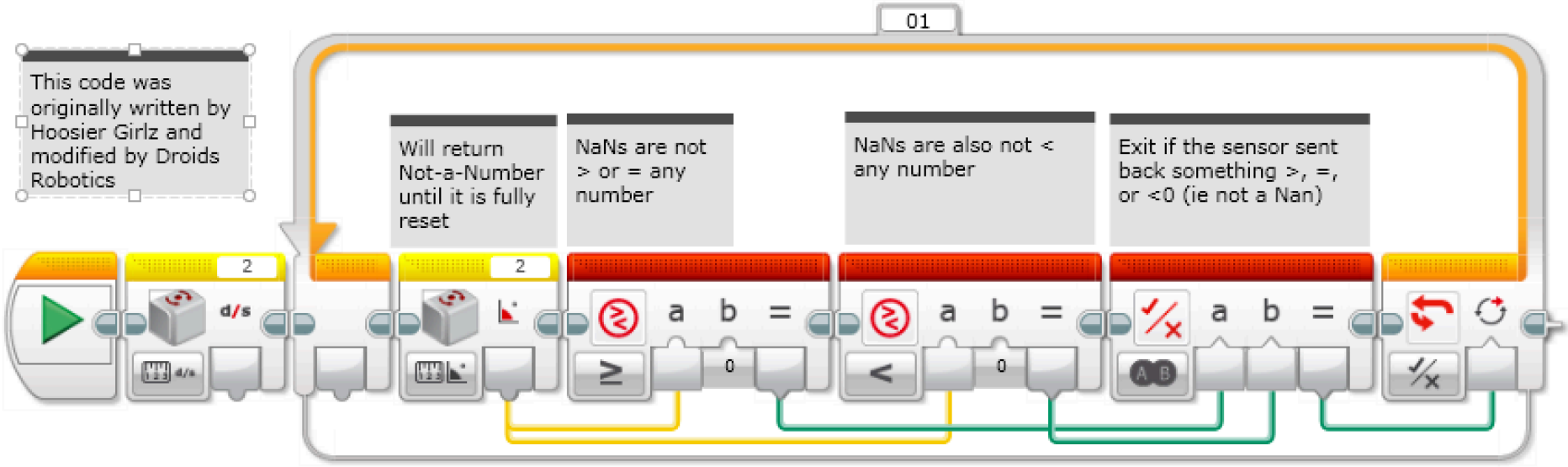
This is a wait block. We wait 3 seconds because it takes time for the gyro sensor to reset to zero. This type of reset takes much longer

Note that in the rest of your program, you should only use the “rate + angle” modes of the gyro. Using the "angle" or “rate” mode will cause the gyro to recalibrate. Also, ***DO NOT*** use the gyro reset - this forces the gyro into angle mode which will cause a long 3 second recalibration.

Strategy 3: Pseudocode

- Having a fixed time wait for the gyro to calibrate may not always work.
- The gyro returns “Not a Number” (NaN) until it has actually reset and NaNs are not >, =, or < any number. This is because they are not numbers
- The only way you can know when it is fully reset is to make sure you are getting back a real number, instead of a Not-a-Number value
  - STEP 1: Recalibrate the gyro
  - STEP 2: start a loop
  - STEP 3: read angle
  - STEP 4: check angle >= 0
  - STEP 5: check angle < 0
  - STEP 6: OR outputs of steps 4 & 5
  - STEP 7: If the output of step 6 is true, exit loop
- At this point, the sensor drift should be gone.

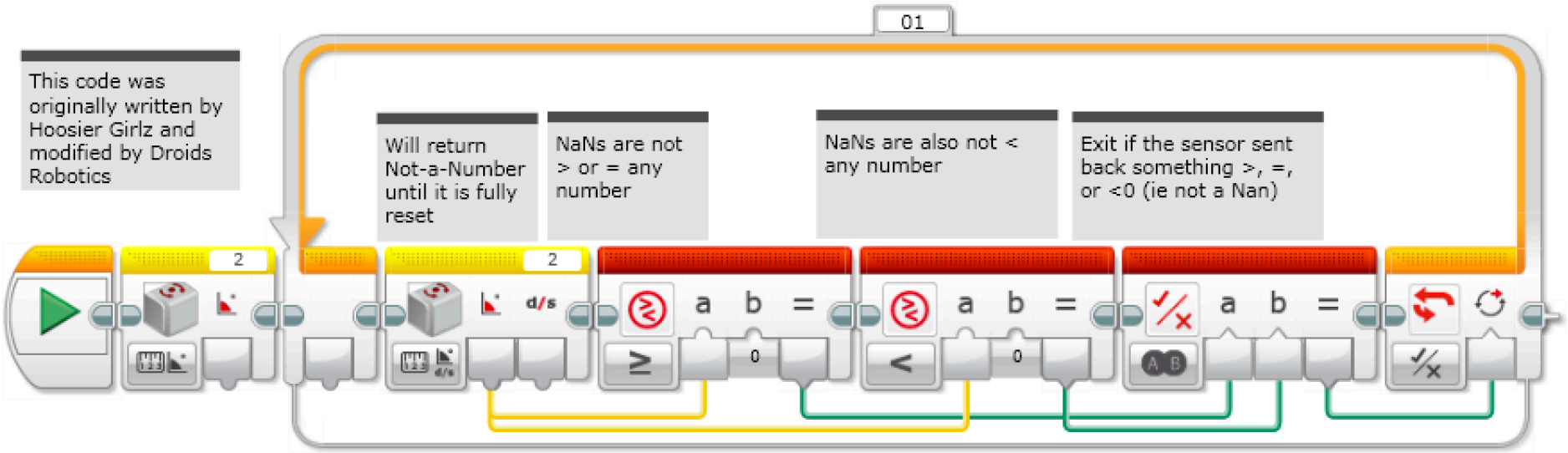
Strategy 3 Solution



This version of the calibration leaves the gyro in angle mode. This is probably the most common way to use the gyro. This code takes about 0.1sec to run.

Note that in the rest of your program, you should only use the "angle" modes of the gyro. Using the "rate" or "rate and angle" mode will cause the gyro to recalibrate.

Strategy 4 Solution



This version of the calibration leaves the gyro in rate+angle mode. This is useful if you use the rate output.

Note that in the rest of your program, you should only use the “rate + angle” modes of the gyro. Using the "angle" or “rate” mode will cause the gyro to recalibrate. Also, ***DO NOT*** use the gyro reset - this forces the gyro into angle mode which will cause a long 3 second recalibration.

Discussion Guide

1. What are 2 common problems when programming with the gyro?

   Ans. Gryo drift and Gyro lag

2. What does Gyro drift mean?

   Ans. The Gyro readings keep changing even when the robot is still

3. Can you move your robot when you calibrate your gyro?

   Ans. No!! Keep the robot still.

4. Do you need to calibrate your gryo before every move?

   Ans. No. Once before you run your entire program

5. Why might it be important to consider multiple solutions to a problem?

   Ans. In robotics, there are multiple ways to solve a problem and there might be tradeoffs between the solutions (e.g. how long the code takes to run the code, can you use both rate and angle readings?)

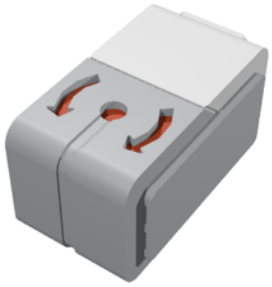
Credits

- This tutorial was written by Sanjay Seshan and Arvind Seshan and uses code shared by Hoosier Girlz (http://www.fllhoosiergirlz.com)
- More lessons at www.ev3lessons.com

MYTHS & TRUTHS ABOUT THE GYRO

By Droids Robotics, 2015

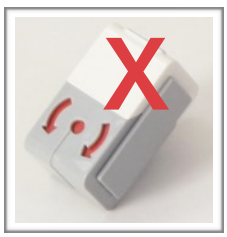
"We used to fear the gyro but we did your @EV3Lessons today at practice and now we love it!" - FLL Team

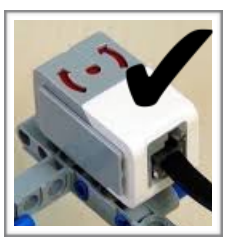


There are numerous myths about the Gyro sensor that we would like to discuss. These myths make teams afraid of trying out the sensor.

The gyro sensor is an extremely useful sensor, but does take a bit of work to use correctly. That is why we have the Gyro lessons in **Advanced** on EV3Lessons.com.

MYTH	TRUTH
The gyro is unreliable for turns.	• The biggest problem with the gyro is drift and lag. Both can be fixed.
You cannot use software to correct for the gyro's drift. All you can do is unplug and replug the sensor.	• There are software solutions you can try. There are several examples of solutions on EV3Lessons.com.
Placement matters: The gyro needs to be low to the ground and at the center of the robot	• See images below. • Where it is on the robot and the height off the ground makes no difference in the readings for FLL. • If the application is for a Gyro Boy or another type of robot that is balancing or has a twisting motion, other installs will work too.
Using two gyros will cancel out the drift.	• Unfortunately, this does not work.
The gyro measures angles	• The gyro measures angular velocity (rate) and computes angle from this.
The gyro cannot be used in FLL reliably	• The gyro can be successfully used in FLL if you correct for lag and drift.
It takes 30secs or more to correct for drift	• Gyro drift takes as little as 0.1 secs and at most 3 secs and is easily done during table set up time in FLL.
Gyro accuracy is an issue	• While the gyro might be a couple of degrees off, other techniques (odometry) can produce similiar or worse errors. Build a robot to tolerate these errors.





Gyro Sensor mounting guide for an FLL robot

1: Angular installs
2: Sideways installs
3: Straight up or down
4: Parallel to ground
5: Upside down, but parallel to ground

ADVANCED EV3 PROGRAMMING LESSON

Gyro Turns

By Sanjay and Arvind Seshan

Lesson Objectives

1. Learn what Gyro Lag is
2. Learn one way to correct for this lag
3. Understand why it is important to explore alternative solutions to a problem

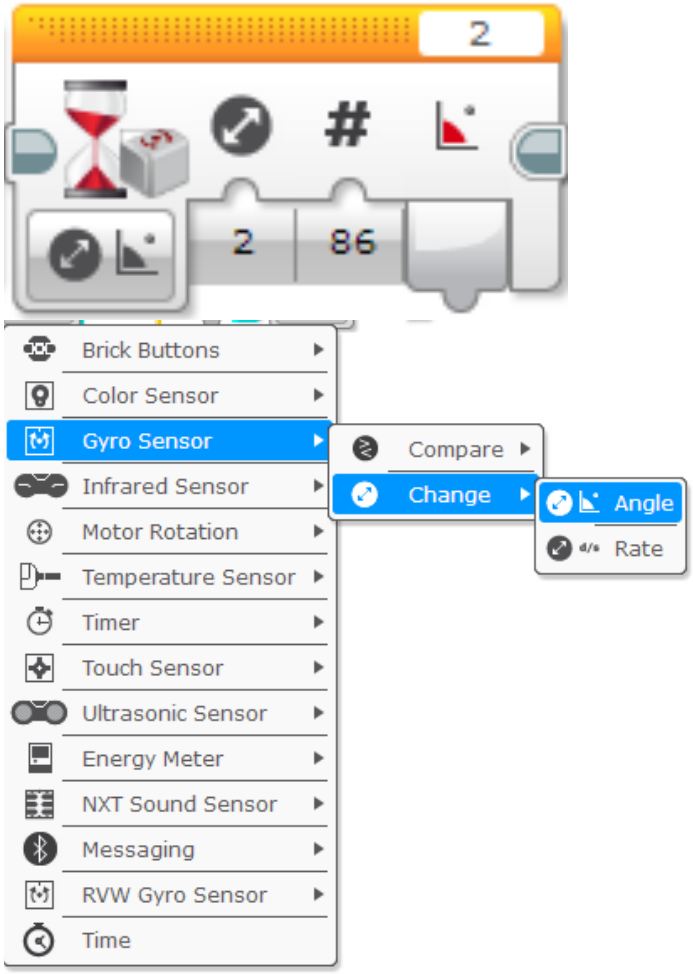
- Pre-requisites: My Blocks with Inputs and Outputs, Data wires, Math Blocks, Loops

Gyro Problem 2: Lag

- What is lag?
  - The gyro sensor readings lag behind the true value sometimes
- When the turn starts, it takes time for the gyro to begin changing
- This lesson presents one way to deal with lag in a turn: reduce the amount of angle that you turn to compensate for lag

Change Mode in Wait Block

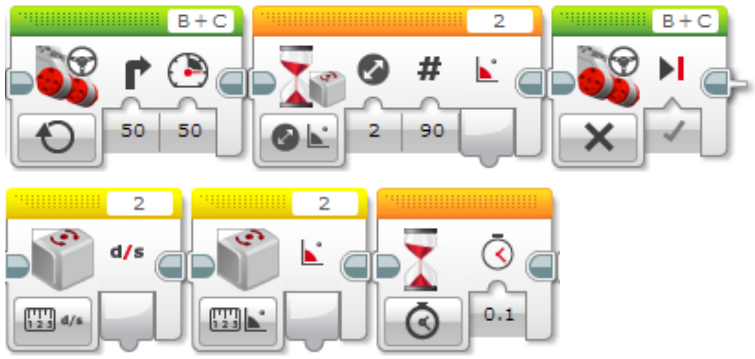
1. In this lesson we use the Wait Block (gyro sensor) in Change Mode
2. Advantages over Compare Mode:
   - You do not need to reset the gyro beforehand
   - You can measure if the value has changed the target degrees by both decreasing or increasing (no need to change the wait block for a left turn)
3. Direction (the first input) defines:
   - 0 – check if the value has increased the desired degrees
   - 1 – check if the value has decreased the desired degrees
   - 2 – check if the value has either increased or decreased the desired degrees



Gyro Turn in Four Easy Steps

STEP 1: Create a simple Gyro Turn program that turns 90 degrees using the Wait for Gyro block in Change Mode

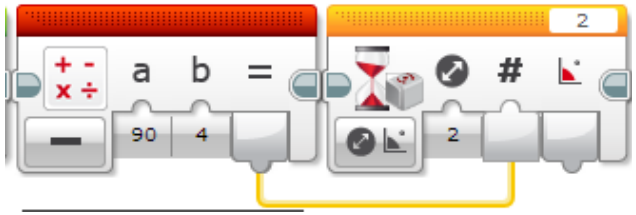
Remember to Calibrate the Gyro before the Wait For Block (see Gyro Lesson for help)



STEP 2: Compensate for Lag

A. Compensate for the lag by reducing the amount of angle to turn based on your robot (e.g 86 degrees instead of 90 degrees)

B. Use a Math Block to create an automatic calculator to compensate for lag



STEP 3: Create and Wire the My Block

STEP 4: Repeat the steps to make one for Left Turns vs. one for Right Turns.

Step 1: Simple Gyro Turn

GOAL OF THE PROGRAM: Simple turn degrees using the gyro

This code is setup for the gyro being connecter to port 2 ; adjust as nedded.

Install tips: The gyro can be anywhere on your robot (even hidden or upside down is okay).

This program turns and waits for the gyro to read 90 degrees. This will make the robot turn 90 degrees to the right.

STEP 1: YOU NEED TO Calibrate THE GYRO: We have these two blocks here because the gyro sensor reading somtimes continues to go up even though the robot is still. By reading the rate of the gryo then the angle the gyro sensor gets recalibrated. Make sure to run these blocks when the robot is still.

We wait 0.1 seconds because it takes time for the gyro sensor to reset to zero.

This block waits until the gyro sensor reads 90.

This stops the motor so it does not keep moving after you reach 90 deg.

Step 2A: Dealing with Lag

Problem with the Step 1: You will find that the gyro does not go the degrees you want it to. If you set it to turn 90 degrees, sometimes it overshoots to 93. You need to make adjustments for your robot because of this. For ours, we needd to turn only 86 degrees in order to turn 90 degrees.

Program goal: A more precise gyro turn

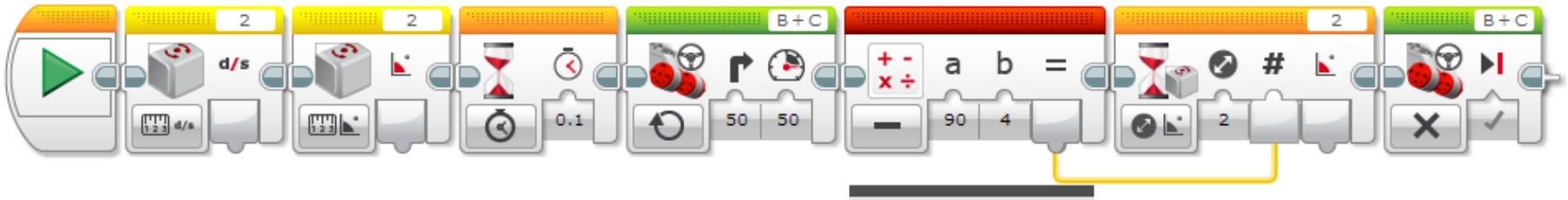
This program turns the robot a bit less than 90 degrees to reach exactly 90 degrees. This value will have to be changed for your robot. The reason the robot does not turn exactly 90 deg. when you type in 90 is because the gyro readings lag behind the robot's actual position.

The only change made from the previous step is that we are now waiting untill the Gyro reaches 86 degrees rather than 90 in this block. This will make a more precise turn

Step 2B: Automatically Correct for Lag

Program goal: subtract degrees automatically

We subtract 4 degrees from your desired degrees using a math block, so we do not need to type in 86 degrees to do a 90 degree turn



This block was added to automatically correct for lag.

The desired degrees is inputed into input a

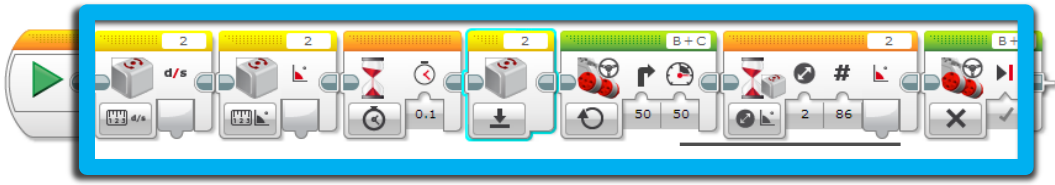
Step 3A: Create a My Block

A. Highlight all the blocks then go to My Block Builder

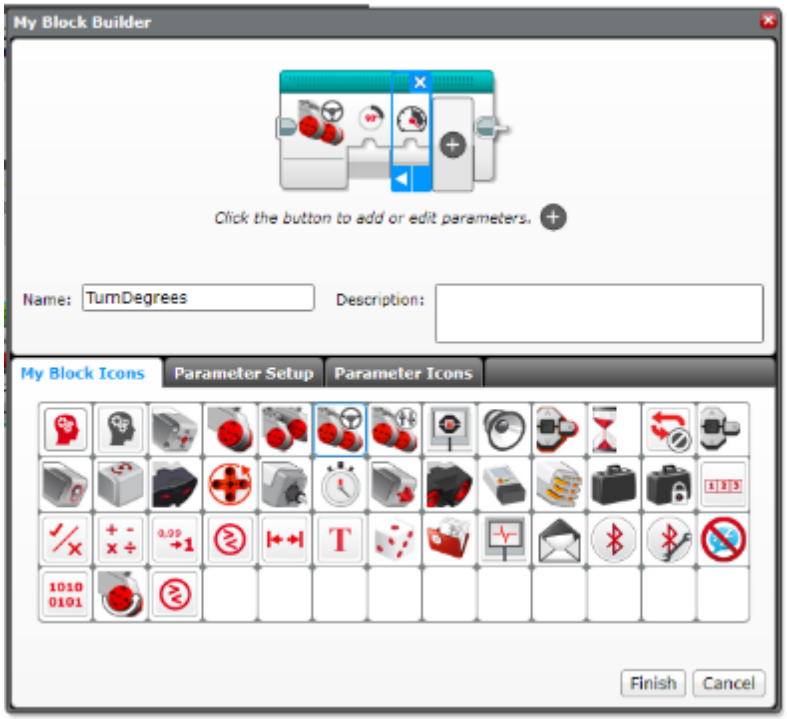
B. Add 2 inputs: one for power and one for and degrees

Refer to the My Blocks with Inputs & Outputs lesson if you need help setting up the My Block

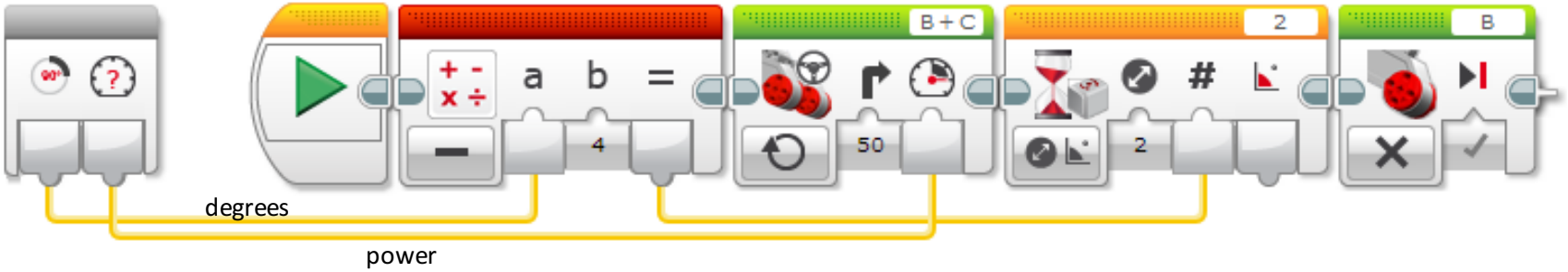
A



B



Stage 3B: Wire the My Block



Connect the degrees value into the math block and the power into the move steering block

Stage 4: Using the My Block

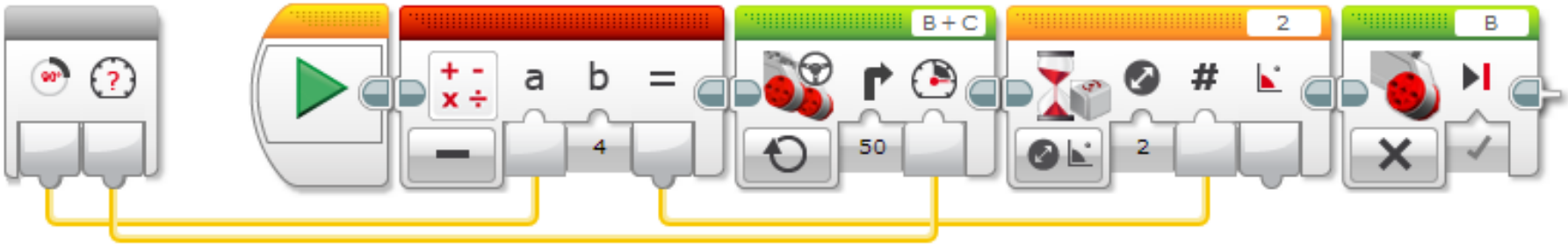
Here is our final stage, it is the same as Step 3, but converted into a my block. It has two inputs, degrees and power. Double click on the my block(s) to see inside.

Here two different my blocks that have been made turn left and right.

DO NOT SELECT the gyro calibrate blocks while making the My Block

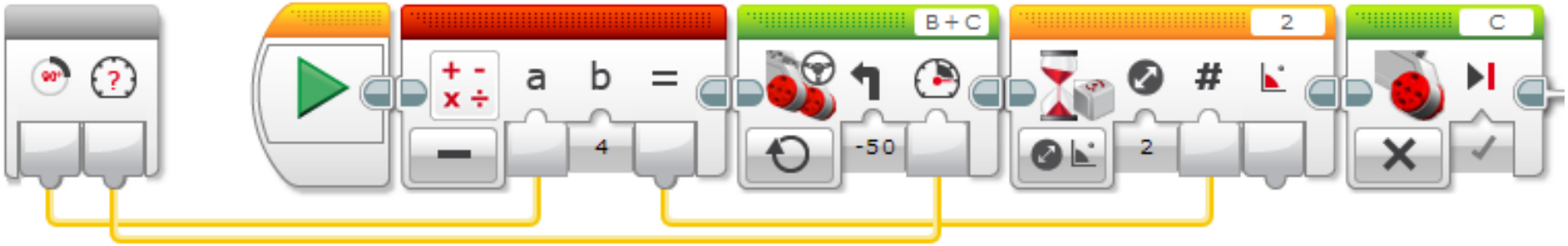
Step 4: Turn Degrees Right

This program is the same as Step 2 other than it is a my block. It was made with two numeric inputs: Power and degrees



Step 4: Turn Degrees Left

This program is the same as Step 2 other than it is a my block. This has been converted to turn left. It was made with two numeric inputs: Power and degrees



Discussion

- What is gyro lag?

  Ans. The gyro sensor's reading lags behind the true reading

- What is one way to compensate for lag?

  Ans: Reduce the number of degrees that you turn

Credits

- This tutorial was written by Sanjay Seshan and Arvind Seshan
- More lessons at www.ev3lessons.com

ADVANCED EV3
PROGRAMMING LESSON

Gyro Move Straight &
Gyro Wall Follow

By Sanjay and Arvind Seshan

Lesson Objectives

- Learn what proportional control means and why to use it
- Learn to apply proportional control to get your robot to move straight
- Lear to apply proportional control to the Gyro sensor to wall follow (move at a particular angle)

- Prerequisites: Math Blocks, Data Wires, Proportional Control, Gyro Sensor

Tips For Success

- You must go through the Proportional Control Lesson and the Proportional Line Follower Lesson before you complete this lesson
- You must also complete the two Gyro Lessons.
- The concept of proportional control is used in this lesson to go straight and wall follow
- Just like for any other proportional control, you need to figure out how to measure error and an appropriate correction
- Video of how the robot will behave: https://youtu.be/0gII2wZs44Y

Pseudocode/Hints

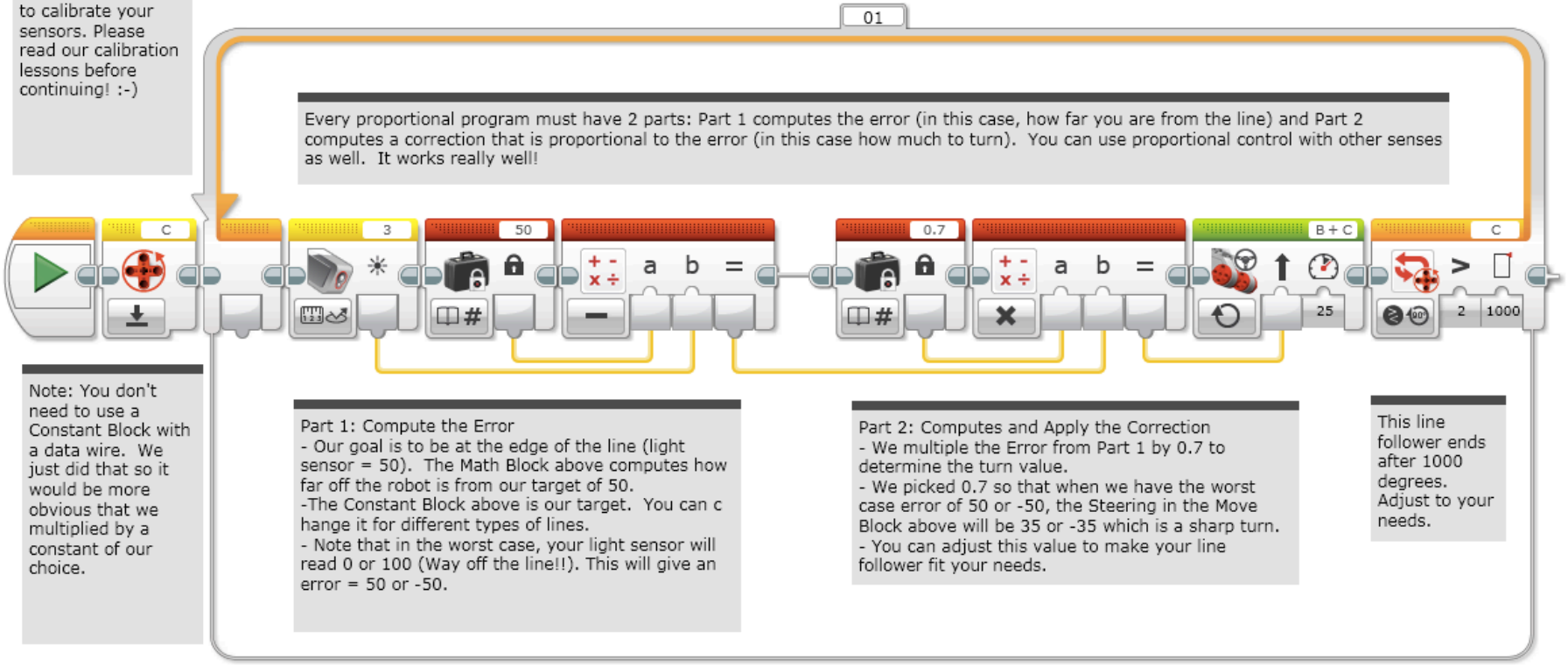
Application	Objective	Error	Correction
Gyro Straight	Make the robot at a constant heading/angle	How far you are from that heading/angle	Turn sharper based on how far you are from that angle
Line Follower	Stay on the edge of the line	How far are our light readings from those at line edge (current_light – target_light)	Turn sharper based on distance from line
Gyro Turn	Turn to a target angle	How many degrees are we from target turn	Turn faster based on degrees remaining

FYI: Proportional Line Follower

Note: This program uses the Color Sensors in Light Mode. This means that you will have to calibrate your sensors. Please read our calibration lessons before continuing! :-)

We recommend that your team uses a proportional line follower like this one. It will be smoothest of the 4 line followers in this lesson. There are even better line followers (that use PID control), but a line follower that uses the "P" is a great start.

A proportional line follower changes the angle of the turn based on how far away from the line the robot is.



Every proportional program must have 2 parts: Part 1 computes the error (in this case, how far you are from the line) and Part 2 computes a correction that is proportional to the error (in this case how much to turn). You can use proportional control with other senses as well. It works really well!

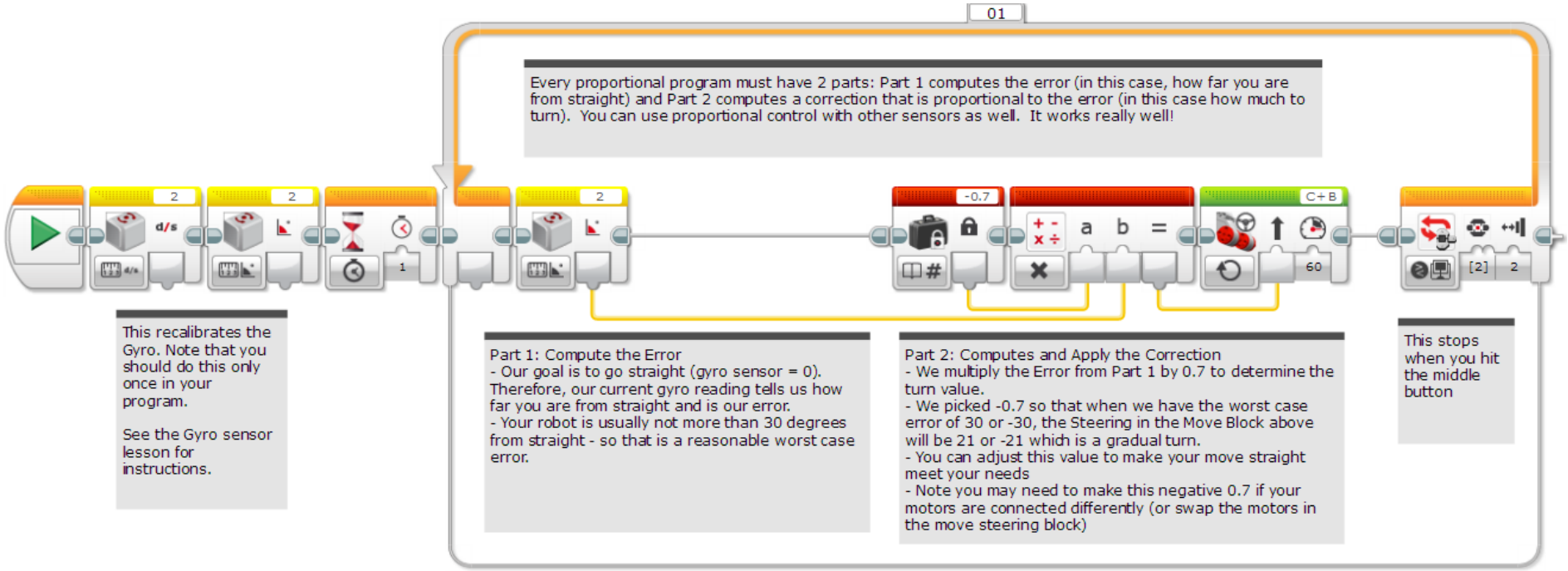
Note: You don't need to use a Constant Block with a data wire. We just did that so it would be more obvious that we multiplied by a constant of our choice.

Part 1: Compute the Error
- Our goal is to be at the edge of the line (light sensor = 50). The Math Block above computes how far off the robot is from our target of 50.
-The Constant Block above is our target. You can c hange it for different types of lines.
- Note that in the worst case, your light sensor will read 0 or 100 (Way off the line!!). This will give an error = 50 or -50.

Part 2: Computes and Apply the Correction
- We multiple the Error from Part 1 by 0.7 to determine the turn value.
- We picked 0.7 so that when we have the worst case error of 50 or -50, the Steering in the Move Block above will be 35 or -35 which is a sharp turn.
- You can adjust this value to make your line follower fit your needs.

This line follower ends after 1000 degrees. Adjust to your needs.

Code: Gyro Move Straight

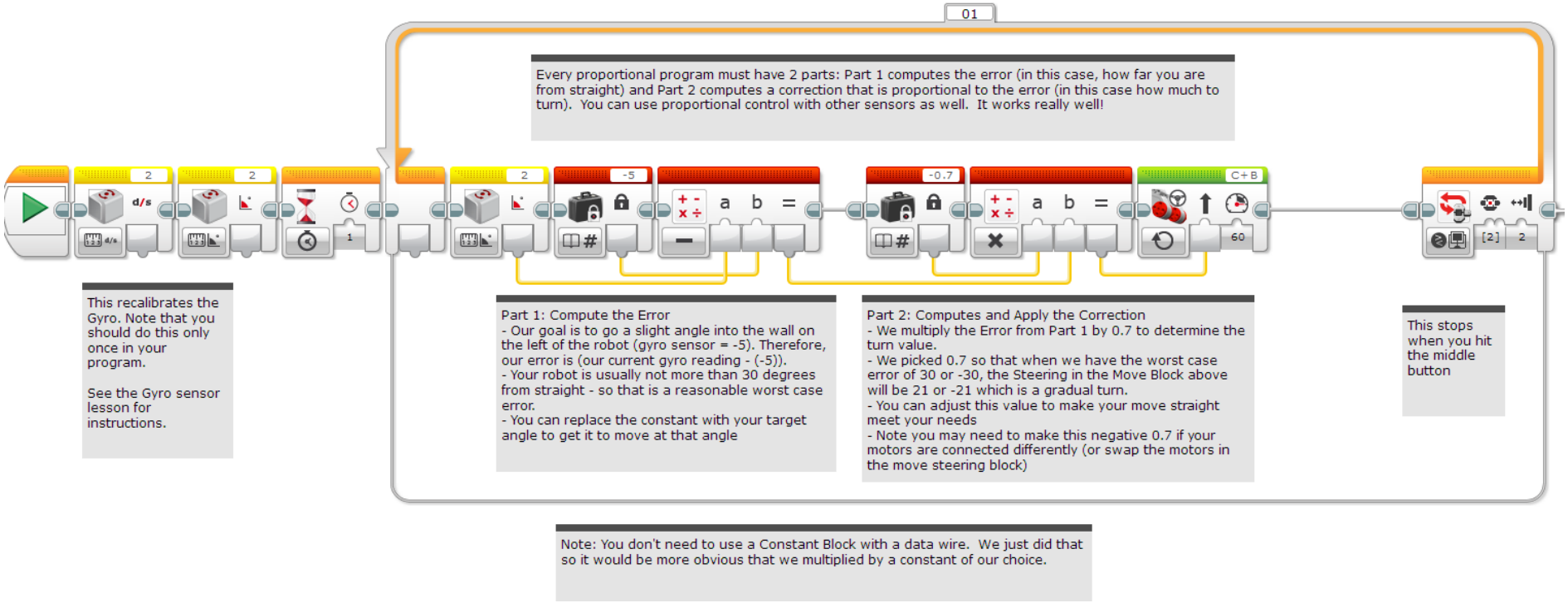


Discussion Guide

1. Compare the proportional line follower code with the proportional move straight code. What similarities and differences do you see?

   Ans. The code is almost the same. The one difference is how the error is calculated. The error is calculated using the gyro sensor. The correction is identical!

Code: Gyro Wall Follow



Discussion Guide

1. Compare the move straight code with the wall follow code. What similarities and differences do you see?
   Ans. There is no target angle for moving straight is 0. But when you want to wall follow, you have to enter a target value of how much you want to angle into the wall.

Credits

- This tutorial was created by Sanjay Seshan and Arvind Seshan
- More lessons at www.ev3lessons.com

INTERMEDIATE PROGRAMMING LESSON

INFRARED SENSOR

By Sanjay and Arvind Seshan

SESHAN BROTHERS

Lesson Objectives

1. Learn how to use an Infrared Sensor
2. Learn to make a remote control system and a program that follows the beacon.
3. Learn to use the Infrared Sensor in all three major modes
4. Learn the limitations of the Infrared Sensor

Prerequisites: Switches, Loops, Compare blocks, and Math blocks

What does the Infrared Sensor do?

Measures proximity to beacon or object

Measures the angle of the beacon relative to the sensor

Measures which button is pressed on remote.

Beacon/remote can be set to 1 of 4 channels. Infrared sensor code must specify which channel to use. This allows you to use multiple remotes in the same room

Infrared Sensor

Beacon/Remote

Three Modes

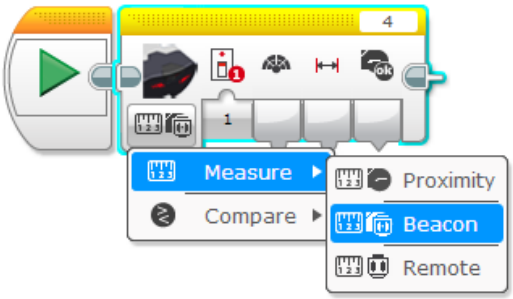
Works up to about 70cm away (or 100 proximity units)

Proximity Mode

- Returns undefined unit type called proximity (not inches or centimeters)

Beacon Mode

- Returns heading (angle) and distance to beacon. Heading measurement is not in degrees.



Remote Mode

- Returns which button is pressed on the remote

We will use all three in this lesson

The Infrared Sensor block can be found in the yellow sensor tab

Challenges

To learn how to use the Infrared Sensor you will complete three challenges:

- Challenge 1: Create a remote control for your robot that does a different action based on which button you press on the Remote
- Challenge 2: Proportional Dog Follower: The robot should move to wherever the Beacon is using proximity and heading
- Challenge 3: Test how accurate the Infrared Sensor is for measuring distances

Pseudocode/Hints

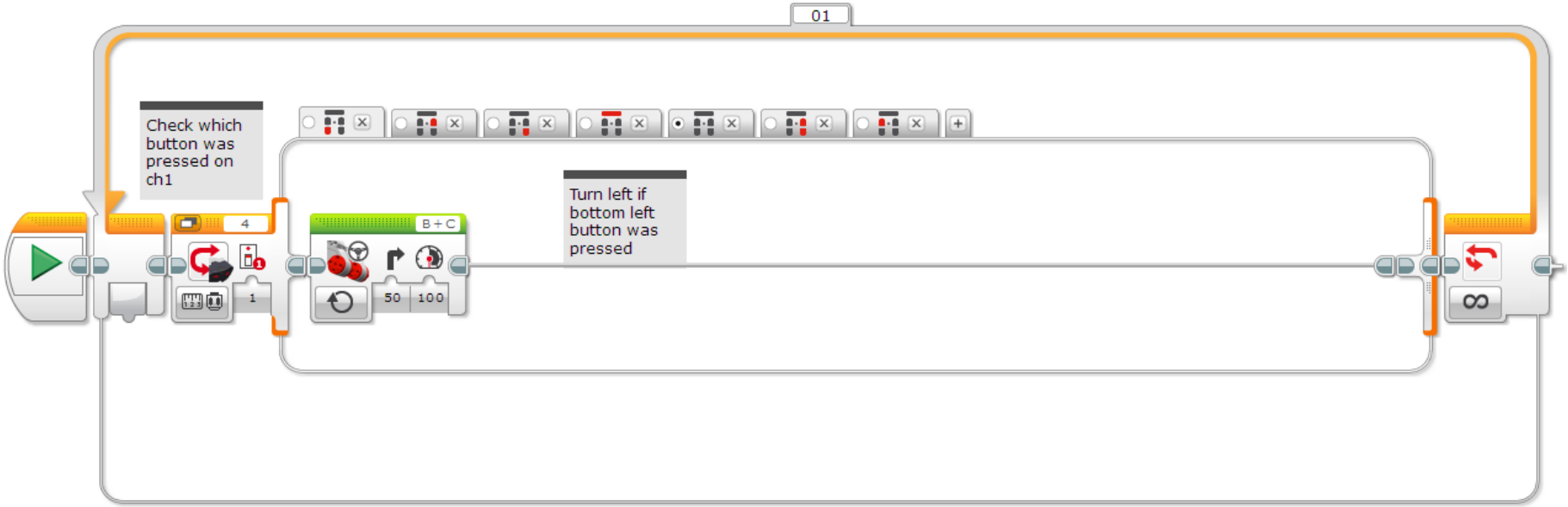
Challenge	Hint/Pseudocode
Remote Control	Run different actions based on which button(s) are pressed on channel 1
Proportional Dog Follower	If the robot is <15 proximity from the beacon move backward If the robot is >15 proximity from the beacon move forward Use proportional control to adjust the steering base on the "heading" of the beacon *Note: Proportional Control is covered in an Advanced Lesson on EV3Lessons.com. Please refer to this lesson.*
Accuracy of Proximity	Measure distance using ultrasonic and measure proximity using infrared (use Port View on brick). Compare measurements for different distances to different surfaces.

Solution: Remote Control

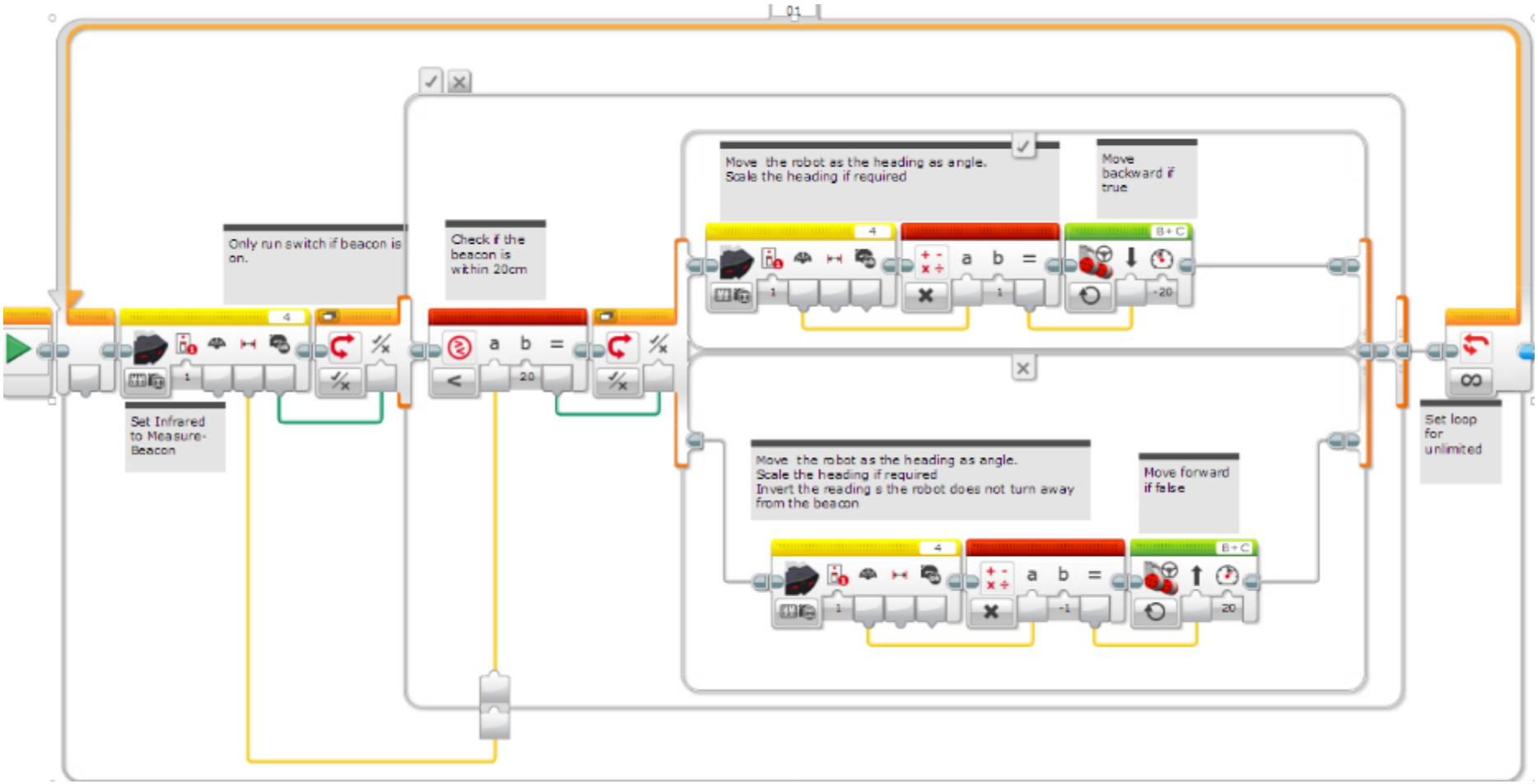
Goal: Create a Remote Control system.
Pseudocode:
1. Check which button was pressed on Ch1 and run a different task based on each button.
2. Repeat forever

Notes: Infrared is connected to port 4 for my robot --- change this for your robot

Make sure to set your remote to channel 1 using the slider button on remote.



Solution: Dog Follower



Challenge 3: Compare Sensors

Surface	Actual Distance to Surface	Ultrasonic Measurement	Infrared Measurement
Aluminum Foil	10CM		
Wooden Table	10CM		
Black Paper	10 CM		
Glass	10 CM		
White Paper	10 CM		

Instructions:

1) Hold the each sensor 10CM away from the material and check the sensor readings on Port View

2) Pick reflective and non-reflective surfaces to try

Lesson:

The Infrared Sensor's reading are based on the intensity of the reflective light. It will not be as accurate as an ultrasonic sensor in measuring how far away an object is. Try different distances next.

Discussion Guide

What modes does the Infrared sensor have?

- Ans: Proximity, Beacon and Remote

Can the Infrared sensor measure distance?

- Yes, but not accurately because it is based on the reflected light intensity. So, it is going to vary based on the material the object is made of.

Next Steps

Go to the Advanced Lesson on the Infrared Sensor (*coming soon)

Read the Advanced Lesson on Proportional Control.

Credits

- This tutorial was created by Sanjay Seshan and Arvind Seshan
- More lessons at www.ev3lessons.com

Droids Robotics Workshop

Intermediate Ohio Workshop

Overview

- Introduction
- Improving Programming Quality – My Blocks
- Improving Robot Reliability - Squaring on a Line
- Improving Line Following - Proportional Control
- Robot Design, Planning, Tools

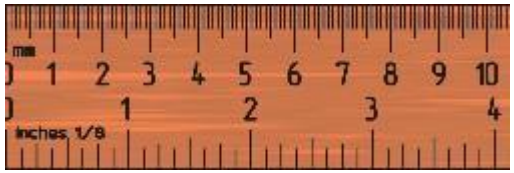
Improving Code Quality

Making a Move Distance My Block (Move_Inches)

By Droids Robotics

Why is a `Move Distance` My Block a good idea

- Built-in move blocks will not take inputs (values) in centimeters or inches.



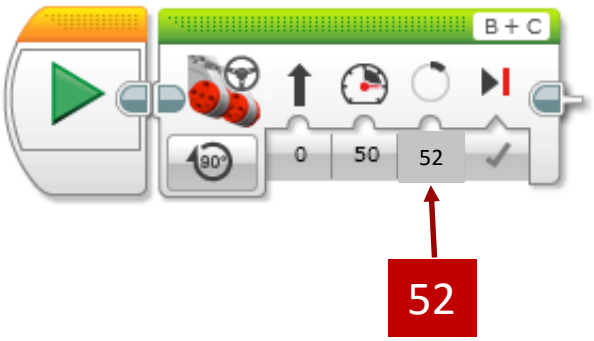
- Much easier to measure distance with a ruler than degrees or rotations.

- If you change your robot design to have bigger or smaller wheels late in the season you don't have to re-measure every movement of your robot
  - Instead of changing distances in every single program you wrote, just go into your cool `Move Distance` Block and change the value for how many inches/cm one motor rotation would take.

Making My Blocks

- During our workshop we showed how to make a My Block with Inputs using the EV3 Software
- Please view our Intermediate EV3Lessons.com lesson on making My Blocks for detailed instructions: PPTX, PDF

PHASE 1: MEASURE WHEELS

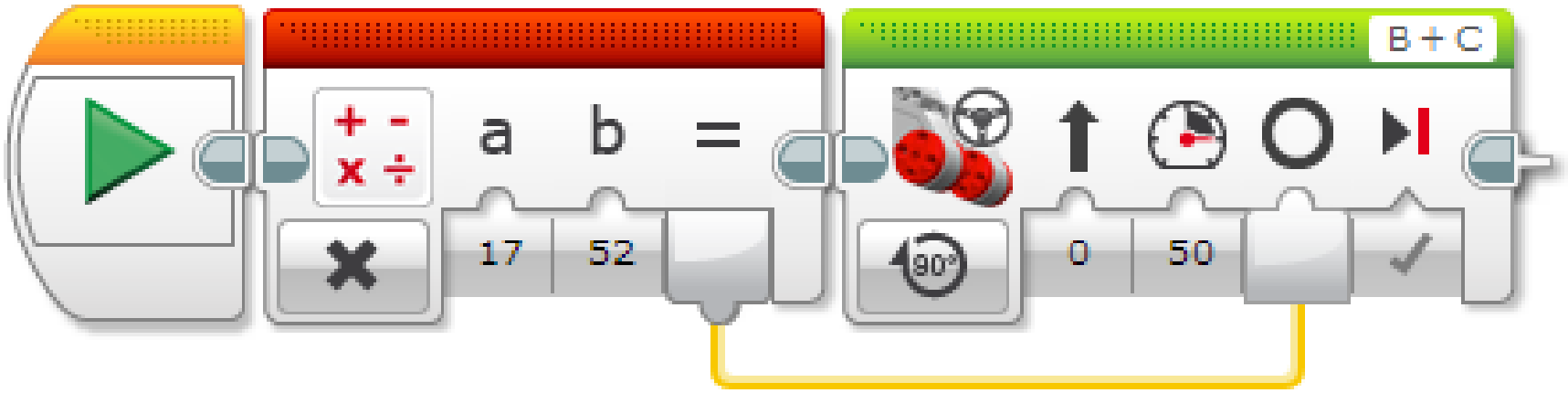


EV3 Base Kit Wheel =
56mm diameter ÷ 25.4 (mm/inch) =
2.2 inch

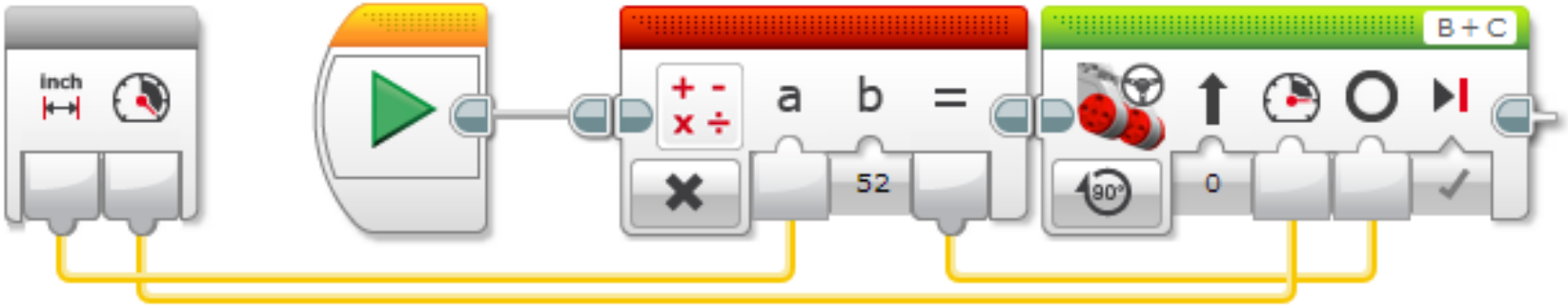
2.2 inches × π = 6.93 inches in each rotation

360 degrees ÷ 6.93 inches = 52 degrees/inch

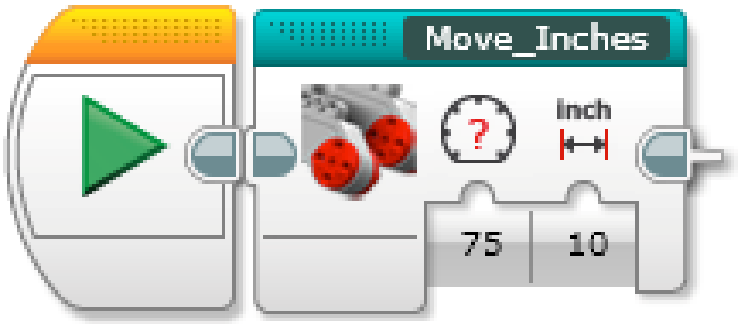
PHASE 2: Math Blocks



Connecting Wires in Move Inches



PHASE 3: COMPLETED Move Inches MY BLOCK



- You can find the completed block in the blue tab at the bottom
- You can double click on the block to view/edit its details

Improving Robot Reliability in FLL

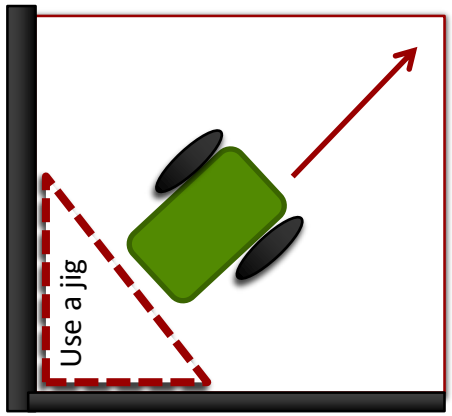
By Droids Robotics

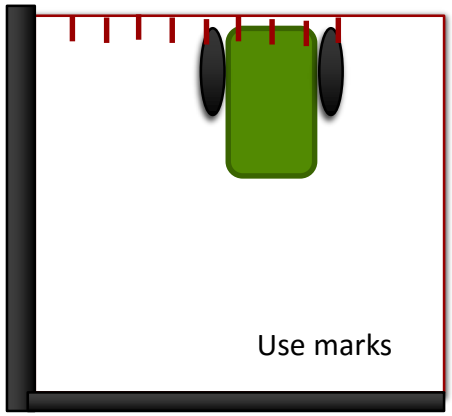
Common Sources of Problems in FLL

Problem	Impact
Alignment in base varies from run to run	Each run is different and missions sometimes work.
Robots don't travel straight or turn exactly the same amount	It is hard to predict the robot location exactly.
Errors accumulate as you travel	Missions far from base fail often. It is hard to do many missions on the same run.

Starting Points in Base are Critical

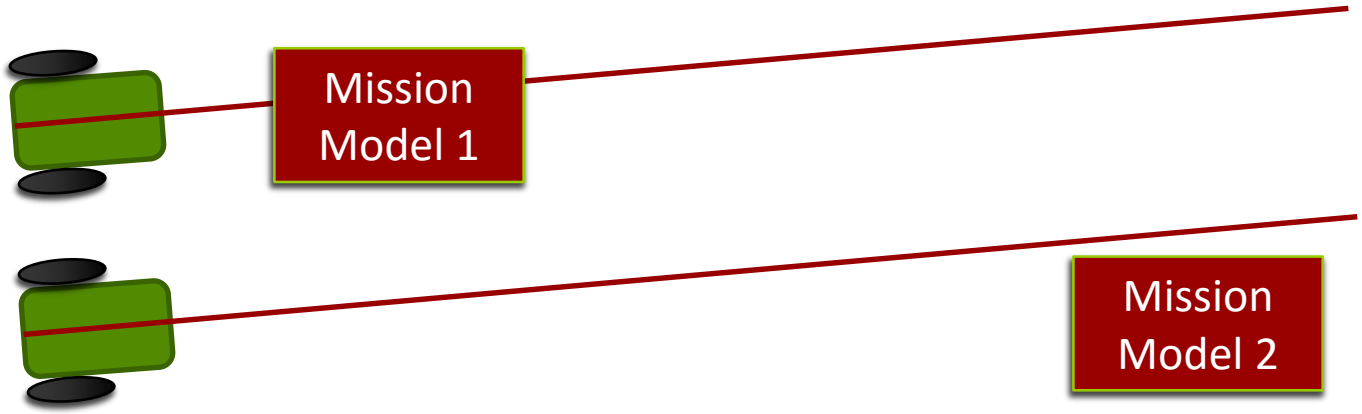
- FLL teams need to figure out where to start in base
  - **Jigs:** a LEGO ruler/wall that your robot can align against them in base
  - **Same start each time:** pick one spot and start there no matter what the mission for easy starts
  - **Tick marks:** Use the inch marks to pick a starting spot for each run
  - **Words:** Base has words. If you aren't near an inch mark, pick a word or letter to start on.
- Even better, try to find a way to align the robot using other techniques





Robot Doesn't Travel Straight & Errors Accumulate Over Time

- By the time you get to the far side of the table, you are no longer in the right position
- Solution: Repeat alignment techniques multiple times in a run for better reliability (see next slide)



Navigation on the Trash Trek Mat

Other Factors in Reliability

- **Battery level**
  - If you program your robot when the battery level is low, it won't run the same when fully charged
    - Motors behave differently with low battery
  - Using sensors makes you not as dependent on battery
- **Motors and sensors don't always match**
  - Some teams test motors, sensors and wheels to make sure that they match
  - You will never get a perfect match so we recommend use other techniques and accept that they will be different

INTERMEDIATE EV3 PROGRAMMING LESSON

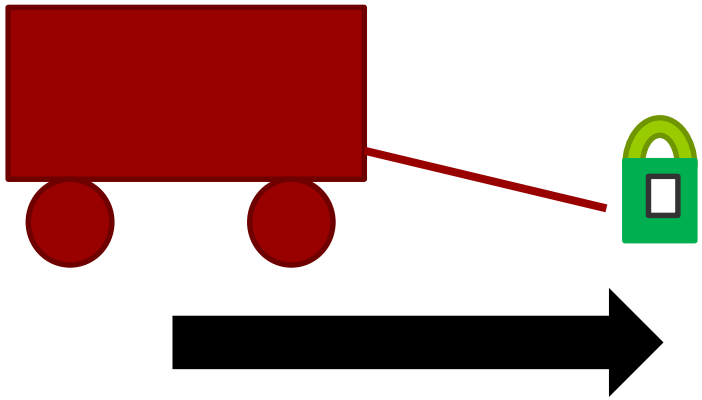
Parallel Beams for Squaring on Lines

By Droids Robotics

What are Parallel Beams?

- Parallel beams allow you to run two or more blocks at the same time.

- In First Lego League, they are mostly often used when you have one of more attachment arms connected to motors and you want to turn these arms while the robot is moving to complete a mission

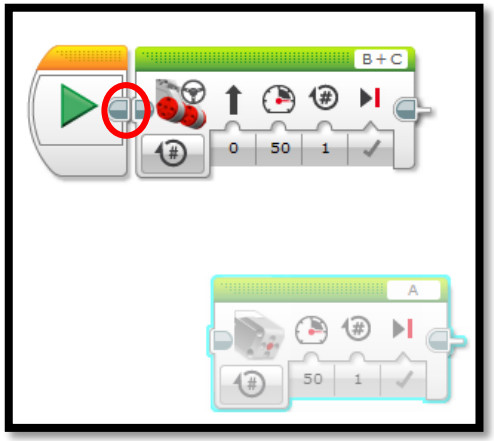


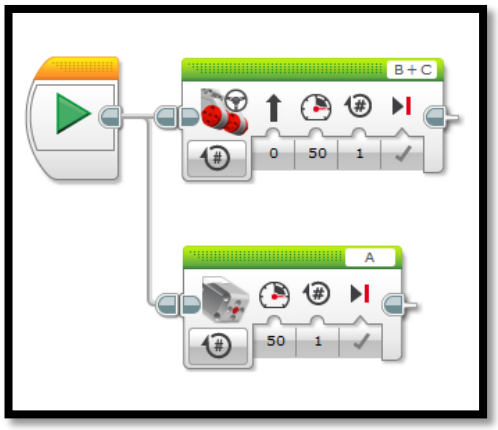
Robot lifting up hoops and driving forward.

How Do I Make a Parallel Beam?

To create a parallel beam click and drag on the bump on the right center of any block and release once you hover over the inverted bump on the left center side on a block.

Note: Blocks before the split will run one at a time. After the split blocks on the two "beams" will run at the same time





Want to learn more?

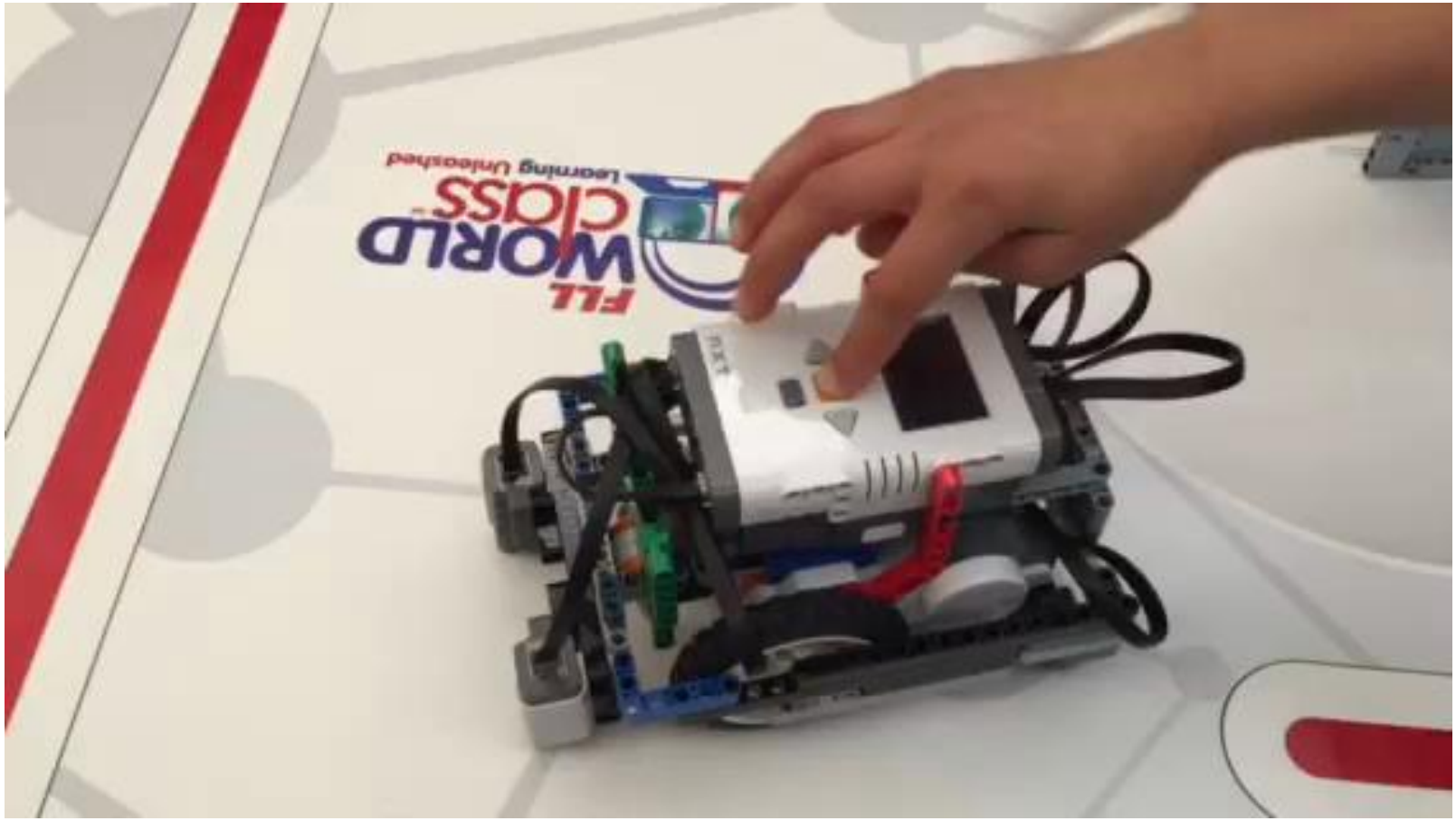
- To learn about Parallel Beams and their limitations/uses, please visit the Intermediate EV3Lessons.com on Parallel Beams.

ADVANCED EV3 PROGRAMMING LESSON

Squaring or Aligning on a Line

By Droids Robotics

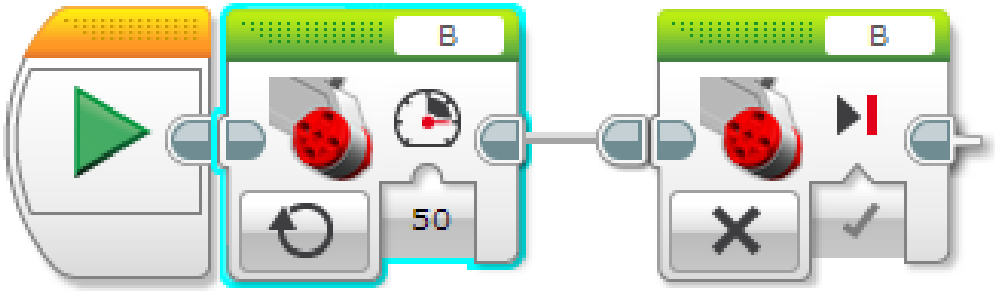
What is Squaring?



Moving Each Motor Independently

- Move Steering lets you control both motors at the same time
- What if you want to move or stop one motor at a time?
  - Use the Large Motor Block

Large Motor Block



Large motor block in ON mode / OFF mode

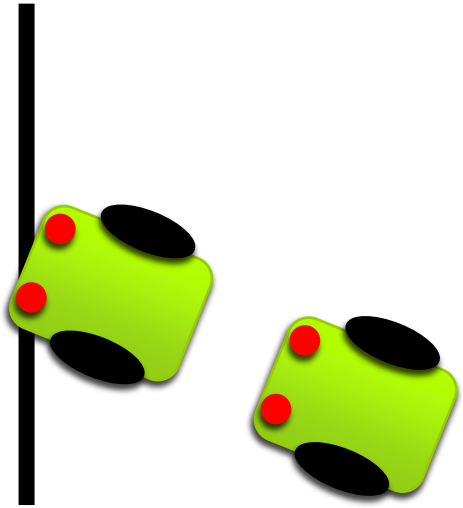
Challenge

Challenge: Make the robot straighten out (align/square off)

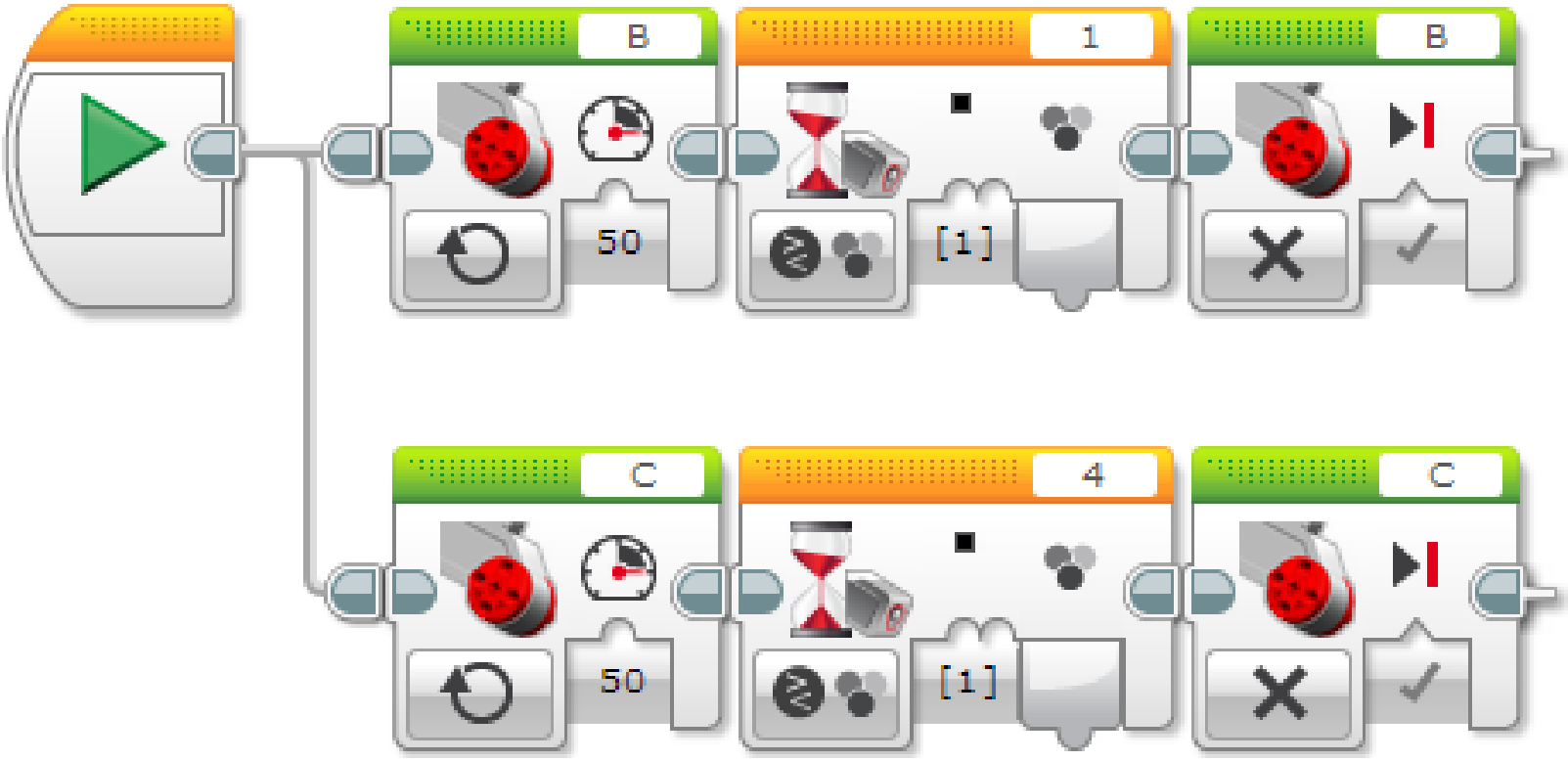
Pseudocode:

1. Start both motors
2. Stop one motor when the sensor on the corresponding side sees the line
3. Stop moving the second motor when the sensor on that side sees the line

Hints: Use a Large Motor Block, Use Parallel Beams



Solution: Align On Line



Repeating a Technique

- What do you notice about the solution we just presented?
  - The robot isn't quite straight (aligned) at the end of it.
  - Both color sensors are on the line, but the robot stops at an angle.
- Challenge Continued: Think about how you can improve this code so that the robot ends straighter
  - Hint: Can you repeat the last process by looking for **white**?
  - This assumes that the line we were straightening out on has white on both sides.

Synchronization errors with parallel beams

- When you have two or more beams you do not know when each beam will finish.
- If you wanted to move after the align finishes you might try to add a move block at the end of one of the beams.
  - Note: This will not work because EV3 code will play your move block without waiting for the other beam to finish.
  - Solution: You need to synchronize your beams. To learn more about synchronization and solutions go to the Advanced EV3Lessons.com Lesson on Sync Beams: PPTX, PDF , EV3 Code
- In this Workshop, we solved the problem of synchronization by making a My Block out of the align code.
  - My Blocks always wait for both beams to finish before exiting
  - You will have 2 inputs: Color and Power

Step 2: My Block With Dual Stage Fix

INTERMEDIATE EV3 PROGRAMMING LESSON

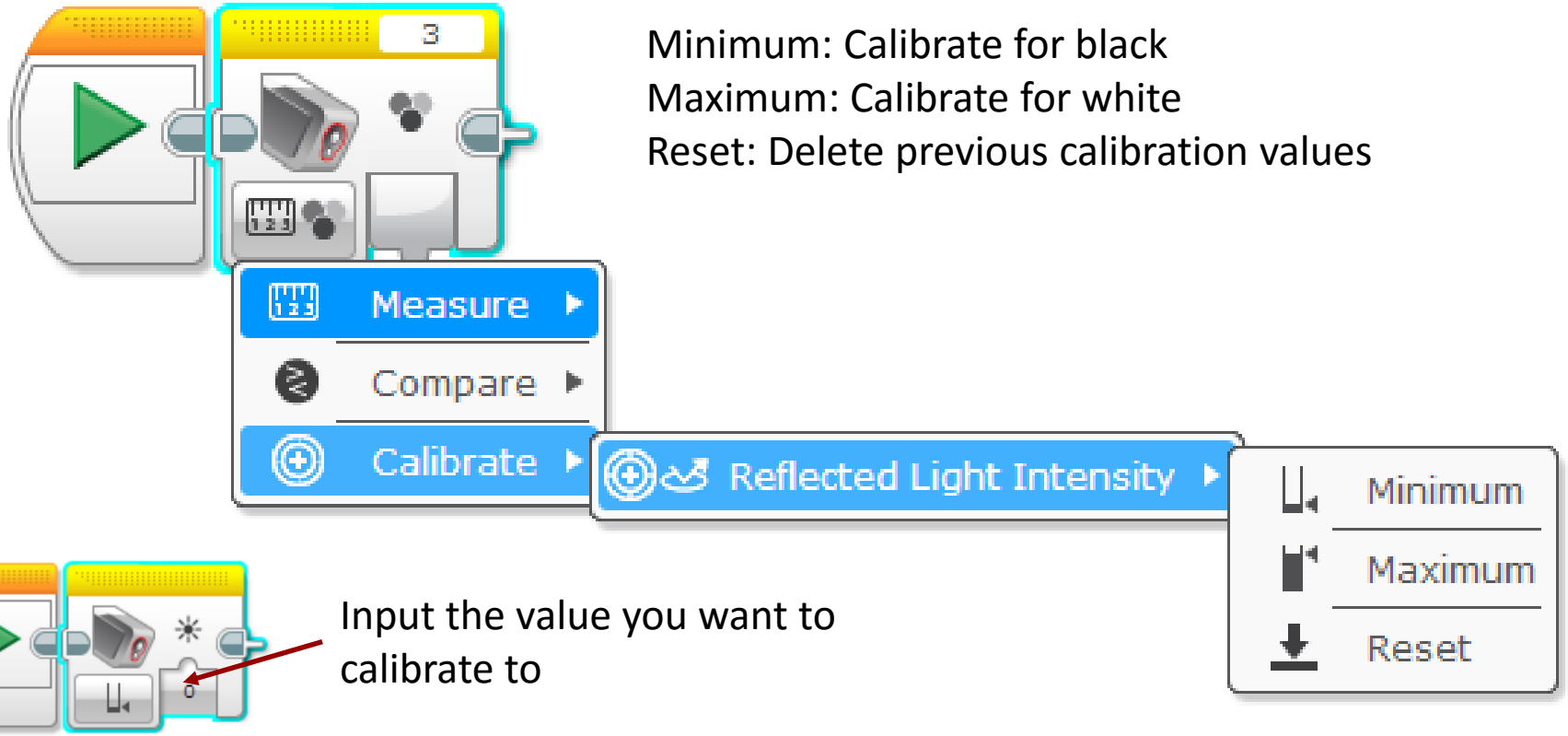
Calibrating Color Sensors for better line followers

By Droids Robotics

Why Calibrate?

- When you use your EV3 Color Sensor in Light Sensor Mode (e.g., reflected light mode), you should calibrate it
- Calibration means “teaching” the sensor what is “Black” and what is “White”
  - This makes White read as 100 and Black read as 0
- Run your Calibrate Program whenever light or table conditions change
- If you are in First Lego League, it is probably a good idea to run it before you start a table run where you use your EV3 Sensors in Light Mode
- If you have 2 Color Sensors, the same calibration will apply to BOTH sensors. You don’t have to make a different calibration program for each color sensor. Make it using 1 sensor on one of the ports and the values will apply to both.
  - If you have sensors that are very different from each other, you will need to write your own custom calibration.

Color Sensor Block: Calibrate Mode



Minimum: Calibrate for black
Maximum: Calibrate for white
Reset: Delete previous calibration values

Input the value you want to calibrate to

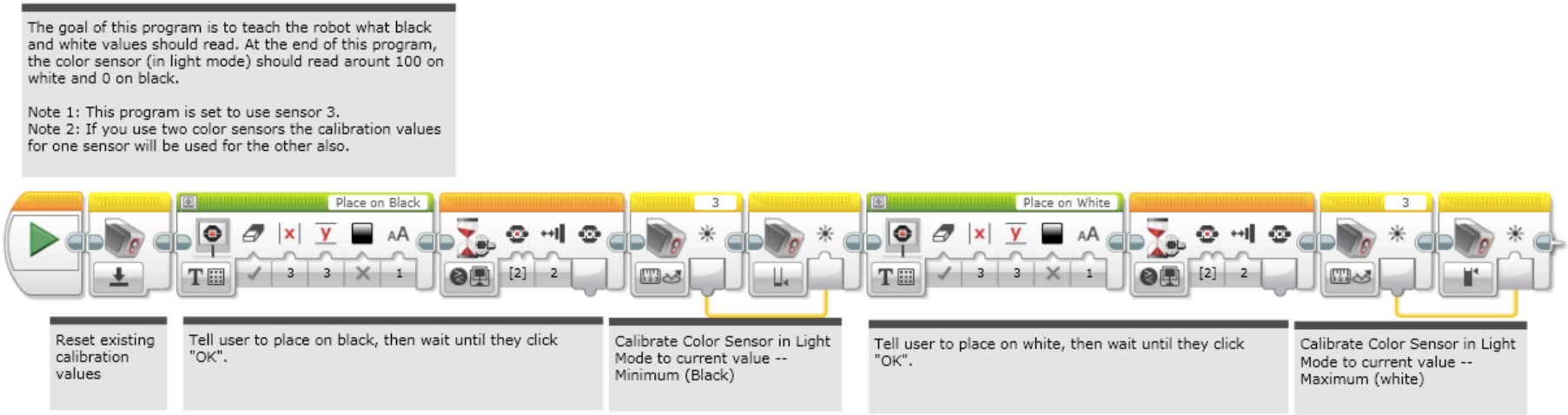
Steps/Pseudocode for Calibration

Challenge: Write a program that will calibrate your EV3 Color Sensors for black and white.

Pseudocode:

- Reset the existing calibration values
- Display that the user should place the robot on “black” and press ok
- Wait for button press
- Read the Color Sensor Block in Light mode and save it to the Color Sensor Block in Calibrate mode.
- Repeat above three steps for calibrating “white”.

Calibrate Program Solution



- When you run the above Calibrate Program, you will be asked to place the robot on a BLACK section of the mat and then hit center EV3 button.
- Then you will be asked to place the robot on WHITE and hit center EV3 button.

ADVANCED EV3 PROGRAMMING LESSON

Proportional Line Follower

By Droids Robotics

Learn and Discuss Proportional Control

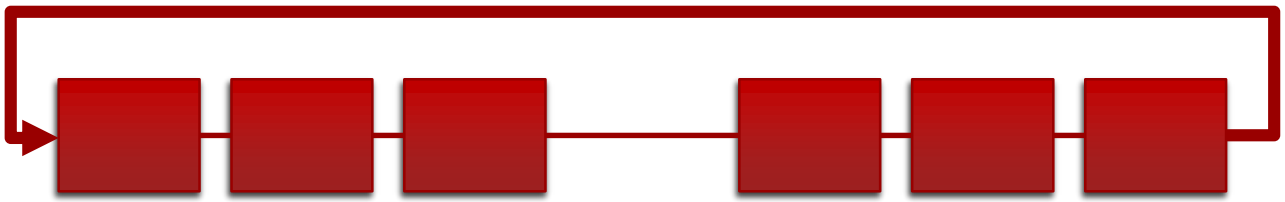
- On our team, we discuss “proportional” as a game.
- Blindfold one teammate. He or She has to get across the room as quickly as they can and stop exactly on a line drawn on the ground (use masking tape to draw a line on the floor).
- The rest of the team has to give the commands.
- When your teammate is far away, the blindfolded person must move fast and take big steps. But as he gets closer to the line, if he keeps running, he will overshoot. So, you have to tell the blindfolded teammate to go slower and take smaller steps.
- You have to program the robot in the same way!

Why Proportional Control?

- What does proportional mean?
  - The robot moves proportionally – moving more or less based on how far the robot is from the target distance
  - For a line follower, the robot may make a sharper turn if it is further away from the line
- Proportional Control can be more accurate and faster

What Proportional Control Looks Like

- The Pseudocode for every proportional control program consists of two stages:
  1. **Computing an error** → how far is the robot from a target
  2. **Making a correction** → make the robot take an action that is proportional to the error (this is why it is called proportional control). You must multiply the error by a scaling factor to determine the correction.

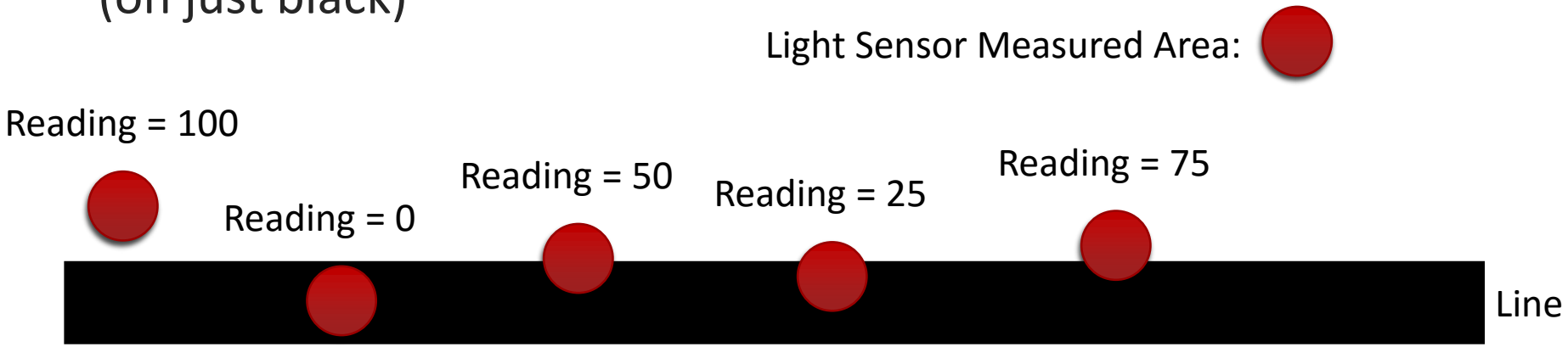


Compute Error

Make Correction

How Far Is the Robot From The Line?

- Reflected light sensor readings show how "dark" the measured area is on average
- Calibrated readings should range from 100 (on just white) to 0 (on just black)



Line Following

- **Computing an error** → how far is the robot from a target
  - Robots follow the edge of line → target should be a sensor reading of 50
  - Error should indicate how far the sensor's value is from a reading of 50
- **Making a correction** → make the robot take an action that is proportional to the error. You must multiply the error by a scaling factor to determine the correction.
  - To follow a line a robot must turn towards the edge of the line
  - The robot must turn more sharply if it is far from a line
  - How do you do this: You must adjust steering input on move block

Pseudocode

Application	Objective	Error	Correction
Line Follower	Stay on the edge of the line	How far are our light readings from those at line edge (current_light – target_light)	Turn sharper (steering) based on distance from line
Gyro Turn	Turn to a target angle	How many degrees are we from target turn	Turn faster based on degrees remaining
Dog Follower	Get to a target distance from wall	How many inches from target location (current_distance – target_distance)	Move faster based on distance

Gyro Turn and Dog Follower are in the Advanced EV3Lessons.com lesson on Proportional Control

Challenge: Proportional Line Follower

Challenge: Can you write a proportional line follower that changes the angle of the turn depending on how far away from the line the robot is?

Pseudocode:

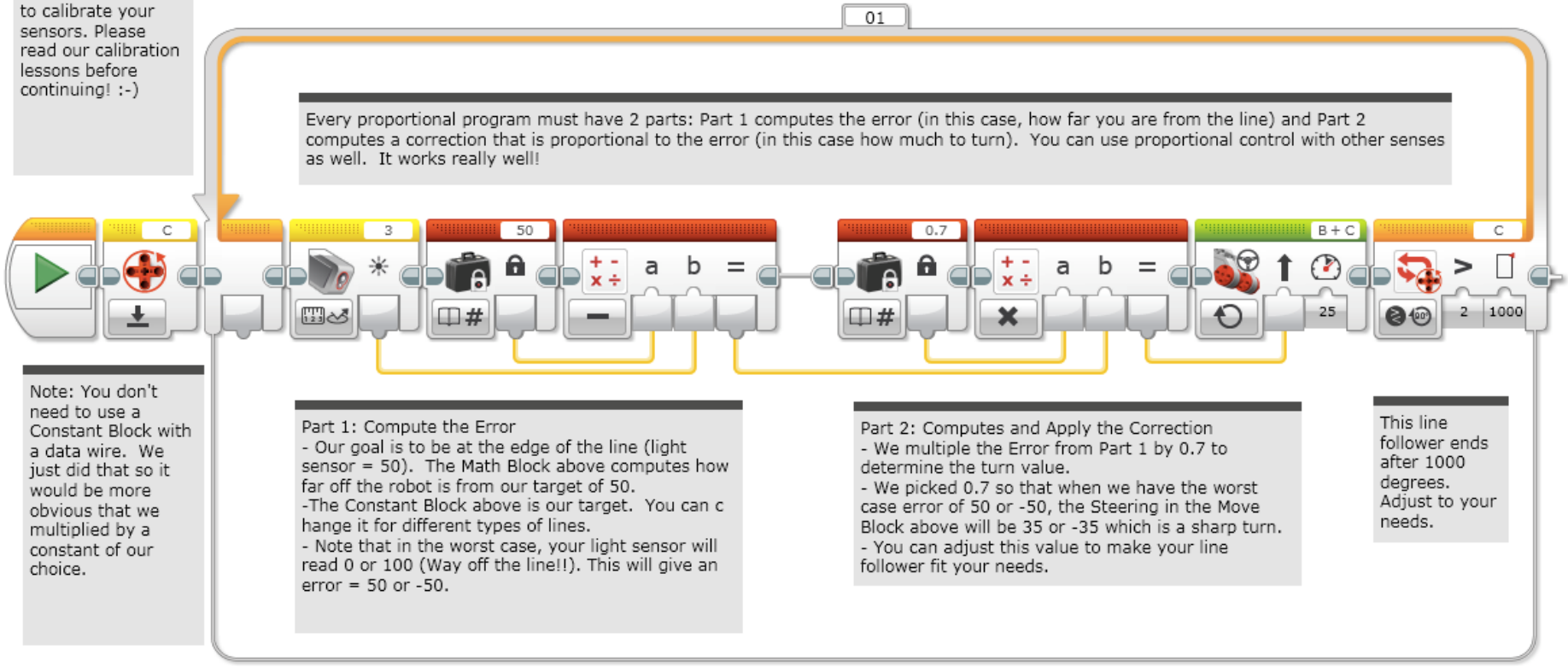
1. Reset the Rotation sensor (Only required for line following for a total distance)
2. Compute the error = Distance from line = (Light sensor reading – Target Reading)
3. Scale the error to determine a correction amount. Adjust your scaling factor to make you robot follow the line more smoothly.
4. Use the Correction value (computed in Step 3) to adjust the robot's turn (steering) towards the line.

Solution: Proportional Line Follower

Note: This program uses the Color Sensors in Light Mode. This means that you will have to calibrate your sensors. Please read our calibration lessons before continuing! :-)

We recommend that your team uses a proportional line follower like this one. It will be smoothest of the 4 line followers in this lesson. There are even better line followers (that use PID control), but a line follower that uses the "P" is a great start.

A proportional line follower changes the angle of the turn based on how far away from the line the robot is.



Every proportional program must have 2 parts: Part 1 computes the error (in this case, how far you are from the line) and Part 2 computes a correction that is proportional to the error (in this case how much to turn). You can use proportional control with other senses as well. It works really well!

Note: You don't need to use a Constant Block with a data wire. We just did that so it would be more obvious that we multiplied by a constant of our choice.

Part 1: Compute the Error
- Our goal is to be at the edge of the line (light sensor = 50). The Math Block above computes how far off the robot is from our target of 50.
-The Constant Block above is our target. You can c hange it for different types of lines.
- Note that in the worst case, your light sensor will read 0 or 100 (Way off the line!!). This will give an error = 50 or -50.

Part 2: Computes and Apply the Correction
- We multiple the Error from Part 1 by 0.7 to determine the turn value.
- We picked 0.7 so that when we have the worst case error of 50 or -50, the Steering in the Move Block above will be 35 or -35 which is a sharp turn.
- You can adjust this value to make your line follower fit your needs.

This line follower ends after 1000 degrees. Adjust to your needs.

Credits

- This tutorial was created by Sanjay Seshan and Arvind Seshan from Droids Robotics.
  - Author's Email: team@droidsrobotics.org
- More lessons at www.ev3lessons.com

ADVANCED EV3 PROGRAMMING LESSON

Line Followers: Basic to Proportional

By Sanjay and Arvind Seshan

Lesson Objectives

- Evaluate and compare different line followers
- Learn to use the concept of “proportional” to create a proportional line follower

- Prerequisites: Basic Line Follower, Color Line Follower, Color Sensor Calibration, Proportional Control, Math Blocks, Data Wires

Which Program Works Best for Which Situation?

Simple Line Follower

- Most basic line follower
- Wiggles a lot due to sharp turns
- Good for rookie teams → need to know loops and switches

Smooth Line Follower

- Almost the same as simple
- Turns are less sharp
- Has trouble on sharp curves
- Good for rookie teams → need to know loops and switches

3-Stage Follower

- Best for straight lines
- Droids do not recommend this. Just learn the proportional line follower.
- Need to know nested switches

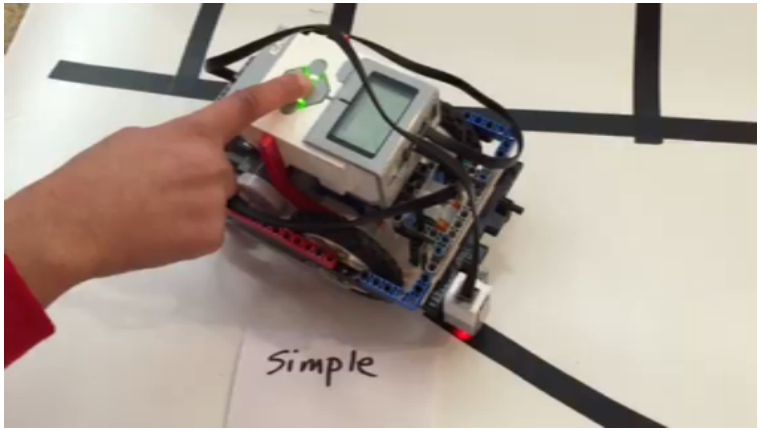
Proportional Follower

- Uses the "P" in PID
- Makes proportional turns
- Works well on both straight and curved lines
- Good for intermediate to advanced teams → need to know math blocks and data wires

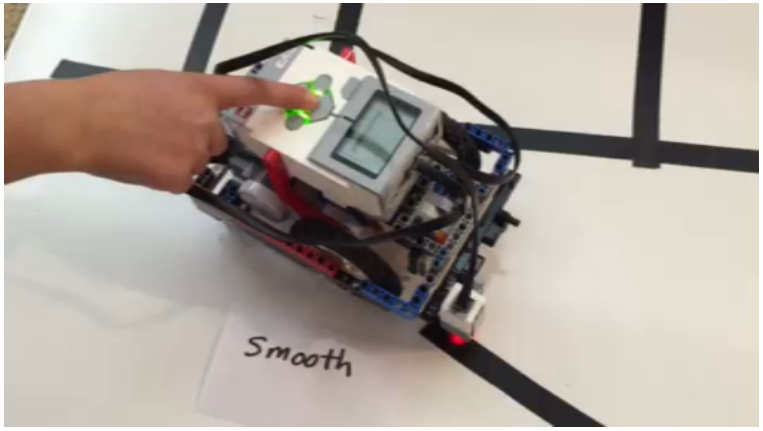
Watch the videos on the next 2 slides to see all four.

Curved Line: Watch Videos

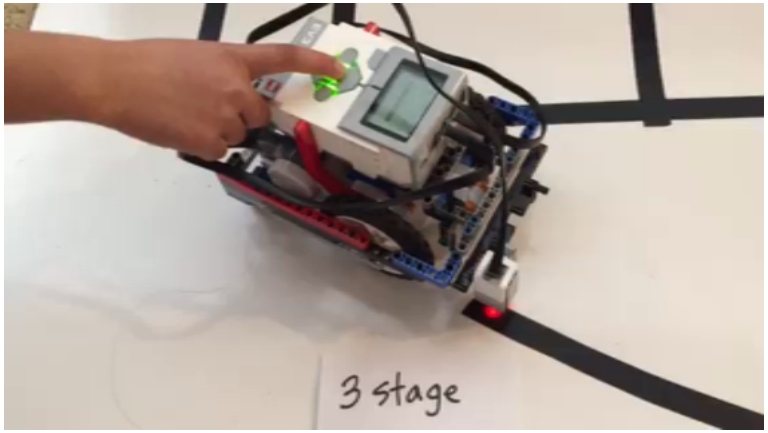
Simple Line Follower



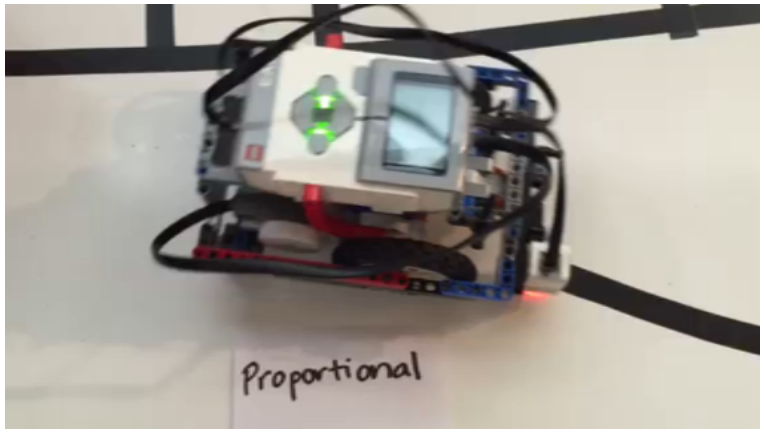
Smooth Line Follower



3-Stage Follower

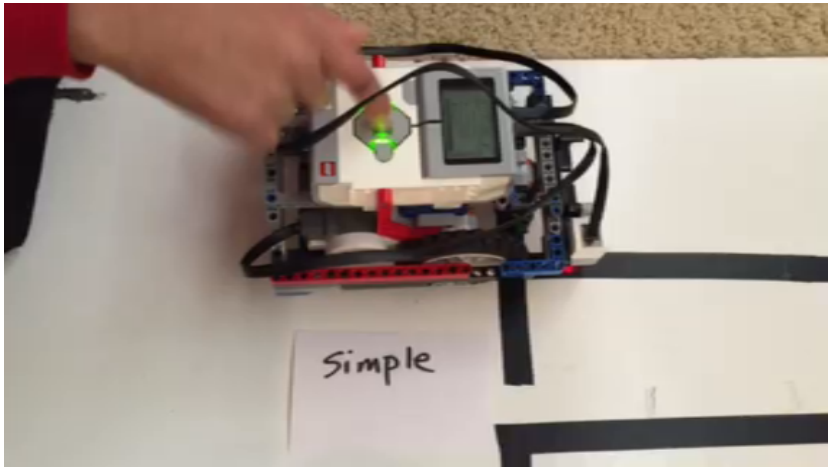


Proportional Follower

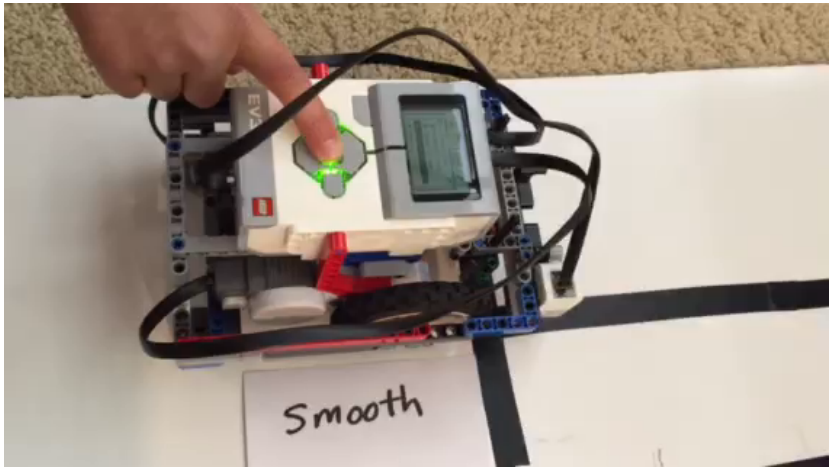


Straight Line: Watch Videos

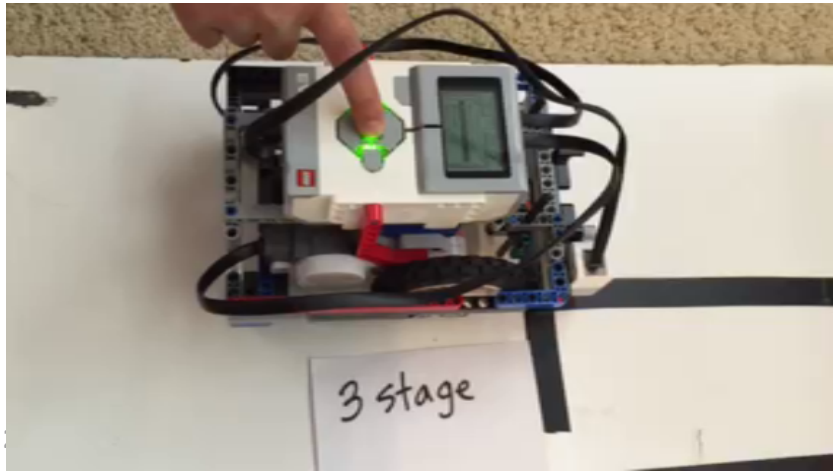
Simple Line Follower



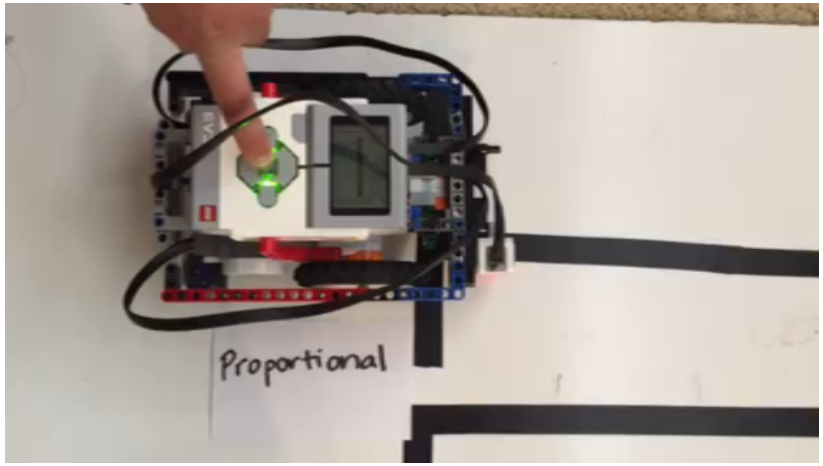
Smooth Line Follower



3-Stage Follower



Proportional Follower



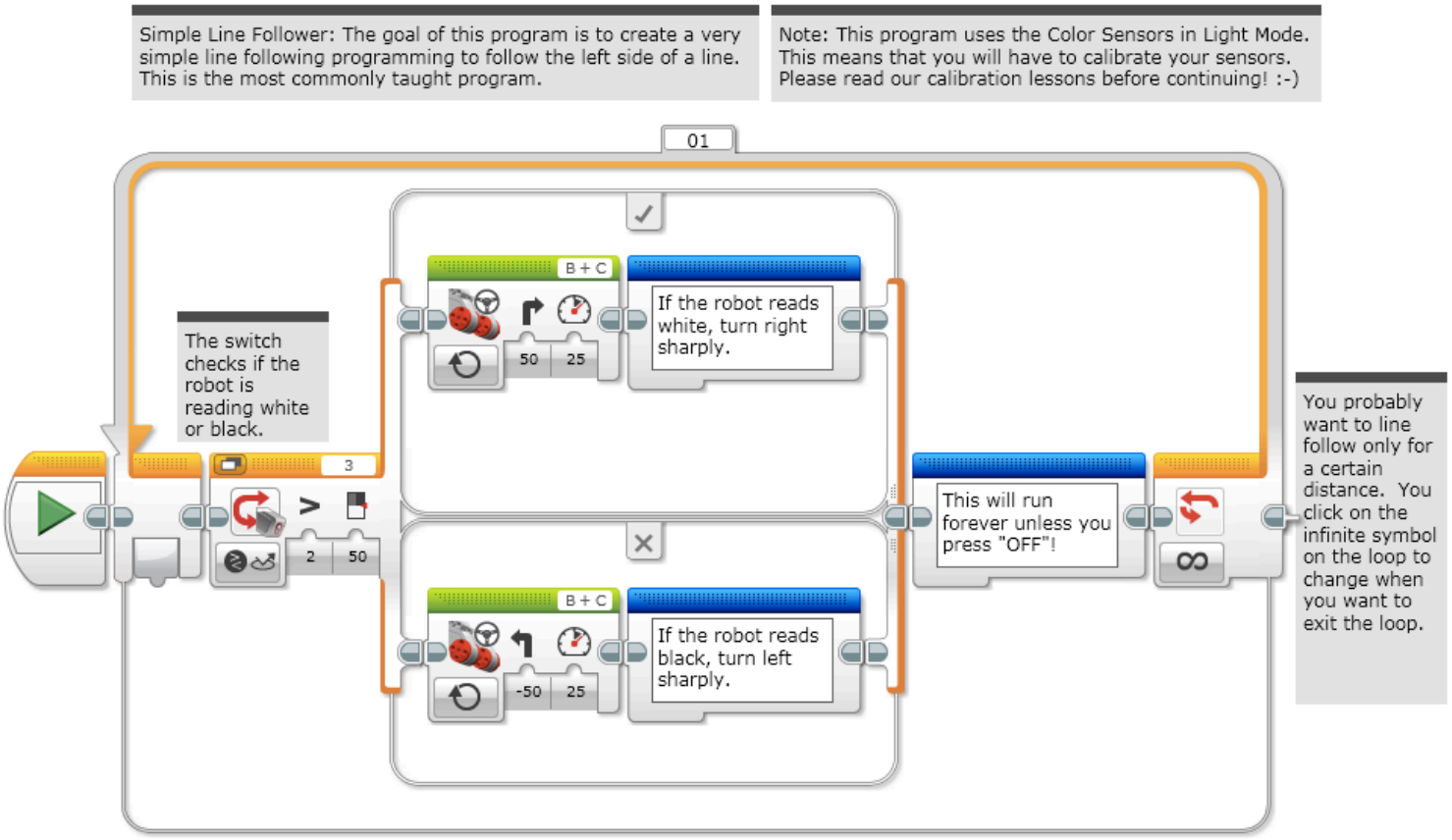
3 Line Follower Challenges

- Challenge 1: Can you write a simple line follower? Hint: Review Beginner: Basic Line Follower lesson
- Challenge 2: Can you write a smoother line follower? Hint: Change how sharp the turns are in a simple line follower.
- Challenge 3: Can you write a three-stage line follower where the robot moves different 3 different ways (left, right or straight) based on the reading from the color sensor?

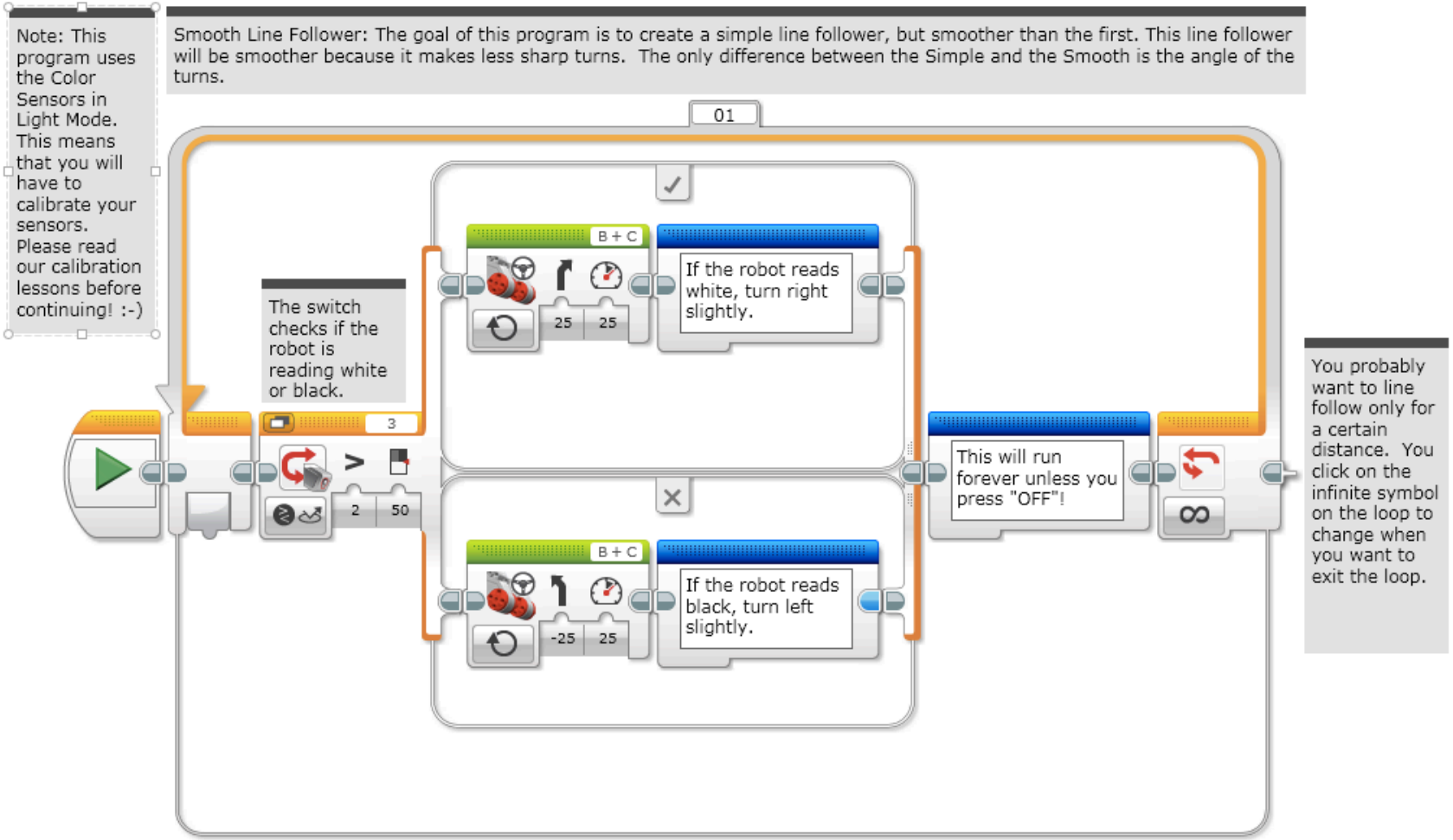
A Note About Our Solutions

- CALIBRATE:
  - The programs use the EV3 Color Sensor in Light Sensor mode
  - You will have to calibrate your sensors.
  - Please refer to Intermediate: Color Sensor Calibration Lesson
- PORTS:
  - The Color Sensor is connected to Port 3.
  - Please change this for your robot.
- WHICH SIDE OF THE LINE:
  - Please take note of which side of the line the code is written for

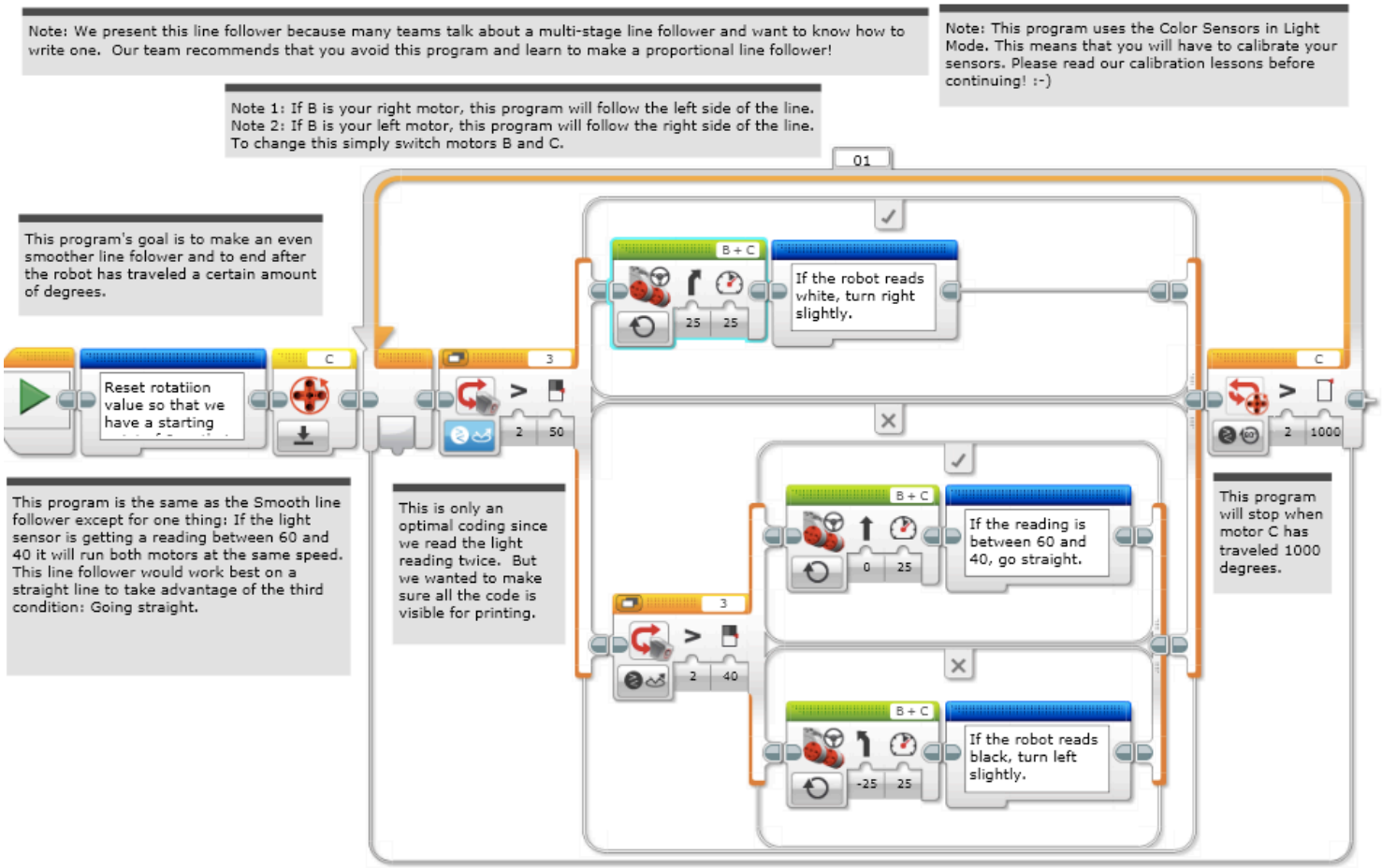
Solution 1: Simple Line Follower



Solution 2: Smooth Line Follower



Solution 3: Three-Stage Line Follower



Challenge 4: Proportional Line Follower

Challenge 4: Can you write a proportional line follower that changes the angle of the turn depending on how far away from the line the robot is?

Pseudocode:

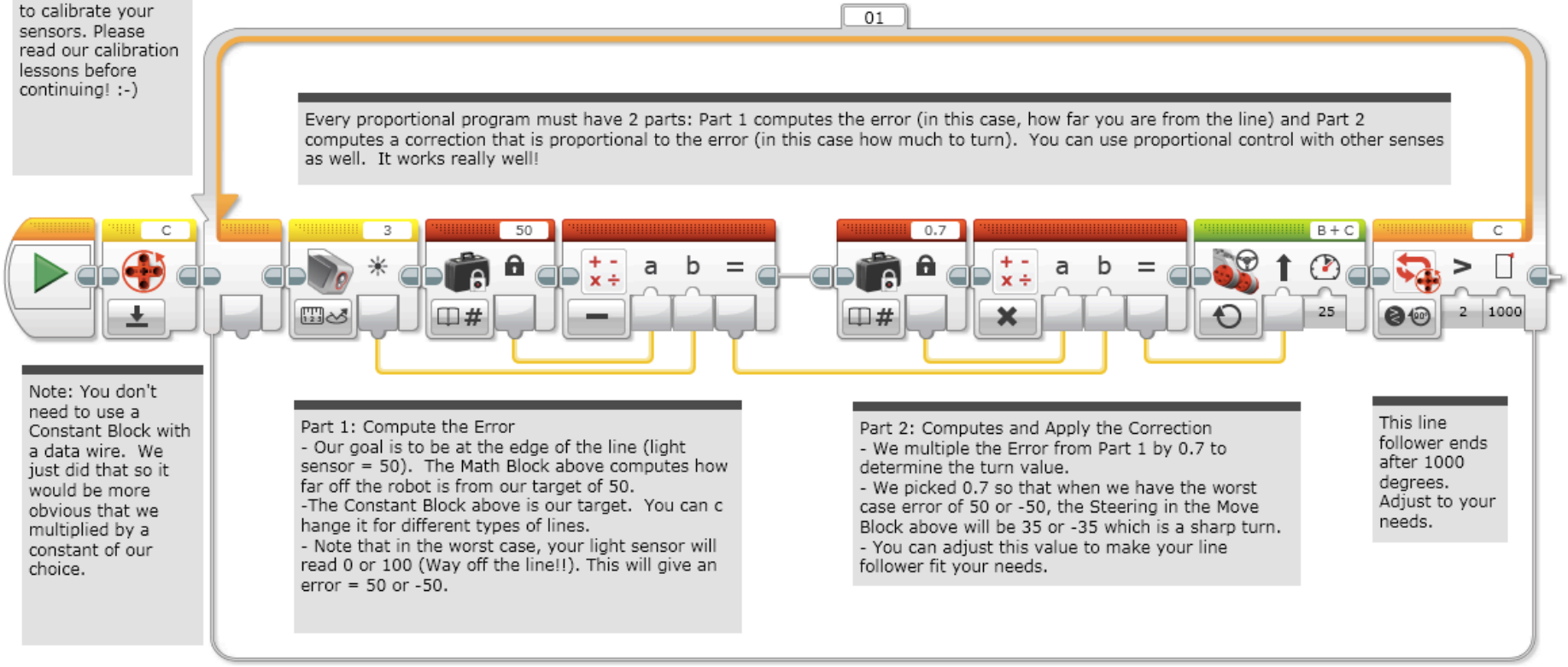
1. Reset the Rotation sensor (Only required for line following for a total distance)
2. Compute the error = Distance from line = (Light sensor reading – Target Reading)
3. Scale the error to determine a correction amount. Adjust your scaling factor to make you robot follow the line more smoothly.
4. Use the Correction value (computer in Step 3) to adjust the robot's turn towards the line.

Solution: Proportional Line Follower

Note: This program uses the Color Sensors in Light Mode. This means that you will have to calibrate your sensors. Please read our calibration lessons before continuing! :-)

We recommend that your team uses a proportional line follower like this one. It will be smoothest of the 4 line followers in this lesson. There are even better line followers (that use PID control), but a line follower that uses the "P" is a great start.

A proportional line follower changes the angle of the turn based on how far away from the line the robot is.



Every proportional program must have 2 parts: Part 1 computes the error (in this case, how far you are from the line) and Part 2 computes a correction that is proportional to the error (in this case how much to turn). You can use proportional control with other senses as well. It works really well!

Note: You don't need to use a Constant Block with a data wire. We just did that so it would be more obvious that we multiplied by a constant of our choice.

Part 1: Compute the Error
- Our goal is to be at the edge of the line (light sensor = 50). The Math Block above computes how far off the robot is from our target of 50.
-The Constant Block above is our target. You can c hange it for different types of lines.
- Note that in the worst case, your light sensor will read 0 or 100 (Way off the line!!). This will give an error = 50 or -50.

Part 2: Computes and Apply the Correction
- We multiple the Error from Part 1 by 0.7 to determine the turn value.
- We picked 0.7 so that when we have the worst case error of 50 or -50, the Steering in the Move Block above will be 35 or -35 which is a sharp turn.
- You can adjust this value to make your line follower fit your needs.

This line follower ends after 1000 degrees. Adjust to your needs.

Tips

- You will get better results
- ….if your color sensors are closer to the ground
- ….remember to calibrate

Discussion Guide

Simple Line Follower

+
+
-
-

Smooth Line Follower

+
+
-
-

Three-Stage Line Follower

+
+
-
-

Proportional Follower

+
+
-
-

Fill in the above with positives and negatives of each technique. Consider if the line follower is best for curved or straight lines. Consider if the robot will wiggle a lot.

Credits

- This tutorial was created by Sanjay Seshan and Arvind Seshan
- More lessons at www.ev3lessons.com

INTERMEDIATE PROGRAMMING LESSON

LOGIC OPERATIONS & DECISION MAKING

By Sanjay and Arvind Seshan

Lesson Objectives

Learn what the Logic Block does

Learn how to use the Logic Block

Prerequisites: Data Wires, Sensor Blocks

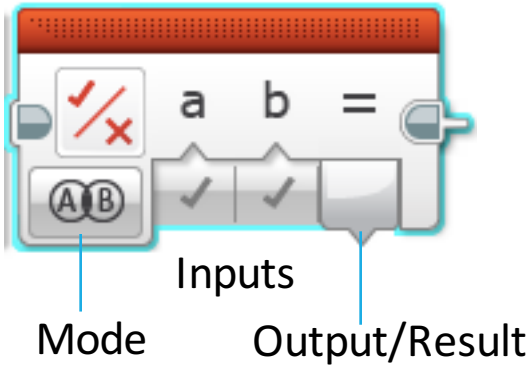
Logic Operations Block

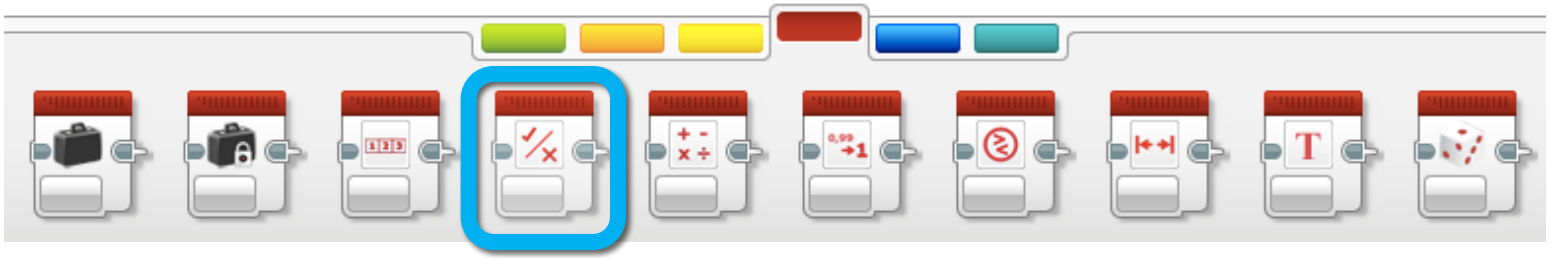
The Logic Bock does a Logic operation on its inputs, and outputs the result

A Logic Block takes inputs that are True or False, and produces a True or False output

Logic values can be used as inputs into loop exists and switch conditions.

It is found in the Red Programming Pallet tab





Different Modes in the Logic Block

Icon	Mode	Inputs	Output/Result
	AND	A, B	• True if both A **and** B are both true, otherwise the result is False
	OR	A, B	• True if either A **or** B (or both) is/are True. The result is False if both A and B are False
	XOR	A, B	• True only if one (and exactly one) of A and B is True • The result is False if both A and B are True • The result is False if both A and B are False
	NOT	A	• Outputs the opposite of what you input. • The result is True if A is False • The result is False if A is True

The icons are Venn Diagrams. The dark shaded areas identify what needs to happen for the block to output True.

Logic Blocks in Three Easy Steps

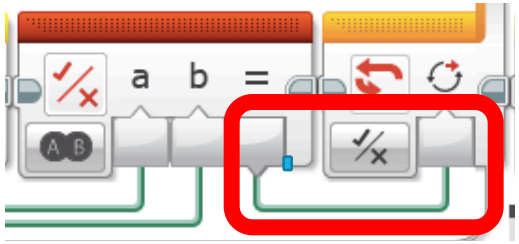
CHALLENGE: Make your robot drive forward until EITHER the Touch Sensor is pressed or the Color Sensor detects black.

STEP 1: Turn the motors on

STEP 2: Add the Logic and Sensor Blocks

- A. Use a Logic Block in the OR mode
- B. Add the inputs: Take a color sensor and a touch sensor blocks and wire them into the Logic Block as inputs

STEP 3: Add a Loop and loop exit condition:

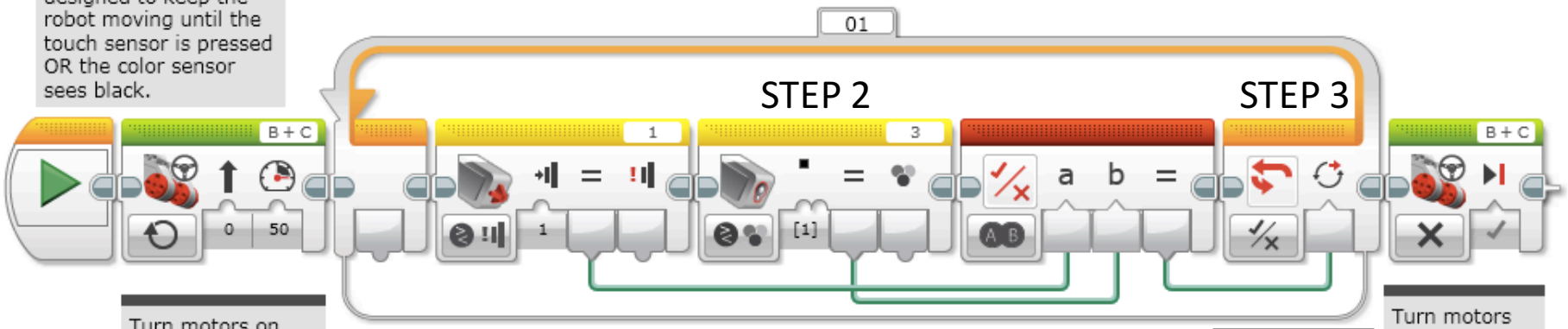


- Place the Sensor and Logic Blocks in a loop
- For the exit condition of the loop, select logic. Wire the result of the Logic Block into the exit condition
- If the result of STEP 2 is True, you should exit the loop and stop the robot

Challenge Solution

STEP 1

This program is designed to keep the robot moving until the touch sensor is pressed OR the color sensor sees black.



Turn motors on

Take the data from the touch sensor and the color sensor and wire them to the inputs in the Logic Block

The Logic Block is in OR mode. The results are wired into the exit condition for the loop.

Exit the loop only when the output of the logic block is True

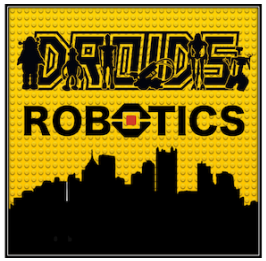
Turn motors off

Credits

- This tutorial was written by Sanjay and Arvind Seshan
- More lessons at www.ev3lessons.com

ADVANCED EV3 PROGRAMMING LESSON

Menu System



By Droids Robotics

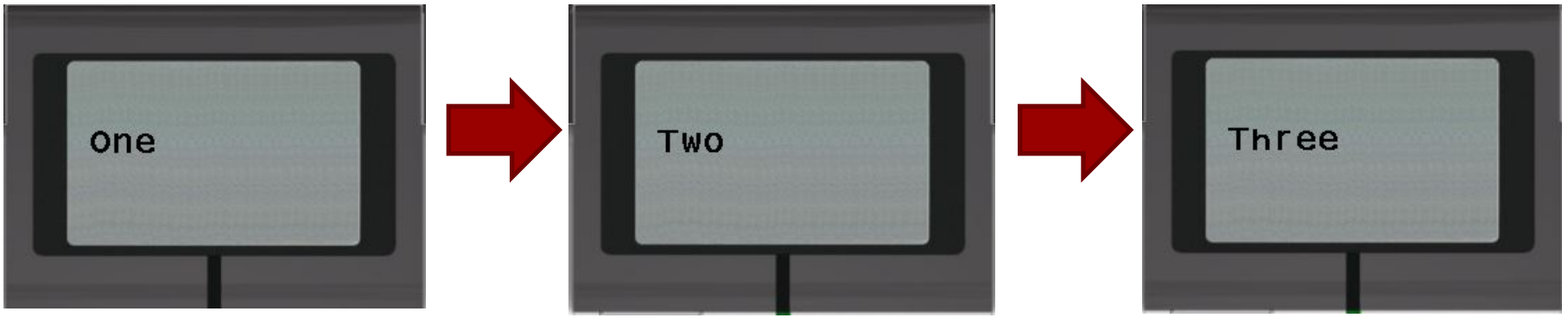
Lesson Objectives

- Learn to use variables
- Learn to create a menu system that is not limited to a particular number of choices
- Learn to create a menu system that updates the menu view

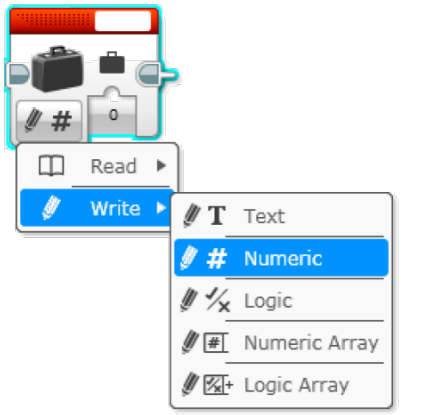
- Prerequisites: Variables, Math Blocks, Brick Buttons

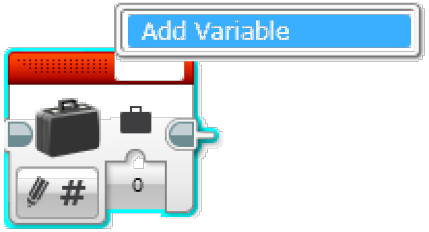
A Fancier Menu System

- In the “Using Brick Buttons as Sensors” Lesson in Intermediate, one of the challenges asked you to create a menu with 4 choices and a single screen display for the entire menu
- In this version, we build a menu system that updates the menu view each time you change your selection and lets you have a larger number of menu choices
- To make this menu, you will need to learn how to use variables



Review: Variables Lessons





A. Variables can store values for later use

B. There are different types of variables. You must chose the type of variable before creating one.

C. You must create a variable and give it a name before using it.

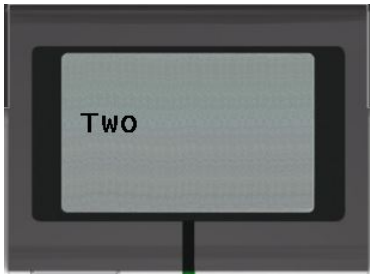
D. Once created, you can read and write values to the variable.

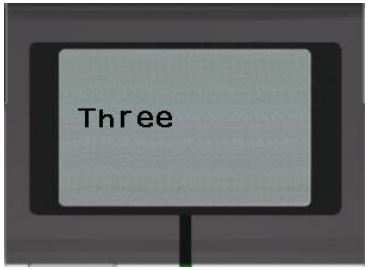
In this lesson, we use numeric variables.

Menu Challenge

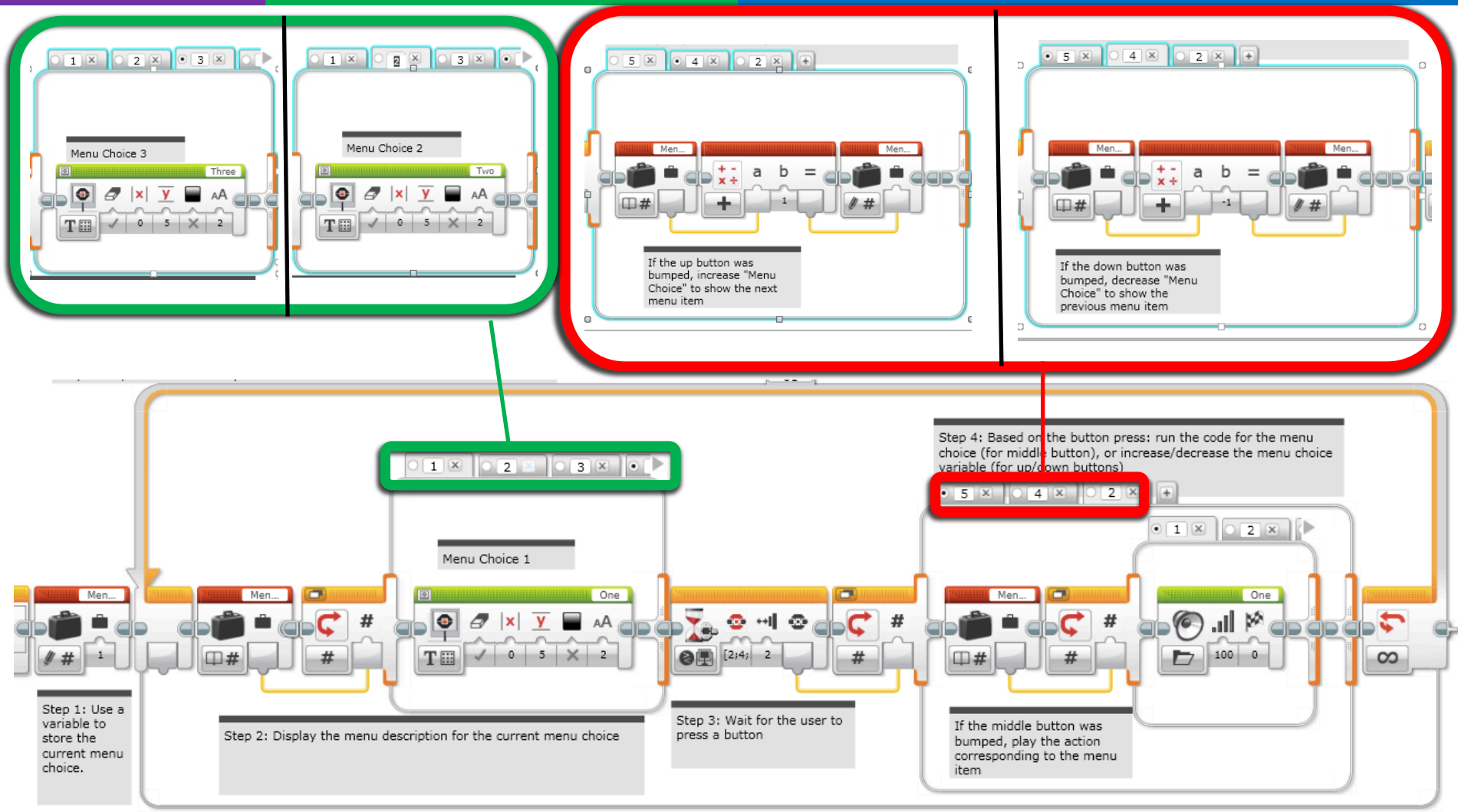
- **Challenge:** Make a menu system that lets you perform 3 actions (display and say the numbers 1, 2, and 3) based on the button pressed
- **Step 1:** Use a variable to store the current menu choice
- **Step 2:** Display the menu description for the current menu choice
- **Step 3:** Wait for the user to press a button (top, middle, down buttons)
- **Step 4:** Based on the button press: run the code for the menu choice (for middle button), or increase/decrease the menu choice variable (for up/down buttons)
- **Step 5:** Go back to 2…

What you will see on the EV3 Brick





Challenge Solution



Next Steps

- The ideas in this lesson can be adapted to help you build a mission sequencer for FLL. Sequencers are useful because they:
  - Allow you to skip missions if you are short of time
  - Allow you to repeat failed missions
  - Allow you access missions quickly (find them easily)
- If your Menu Action code is long (not just a display and sound), consider creating My Blocks out of your code

Credits

- This tutorial was created by Sanjay Seshan and Arvind Seshan from Droids Robotics.
  - Author's Email: team@droidsrobotics.org
- More lessons at www.ev3lessons.com