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

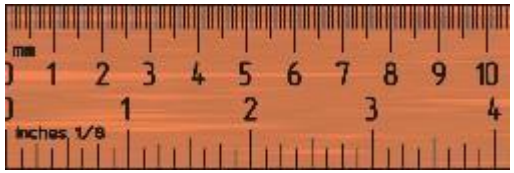
Improving Code Quality

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.



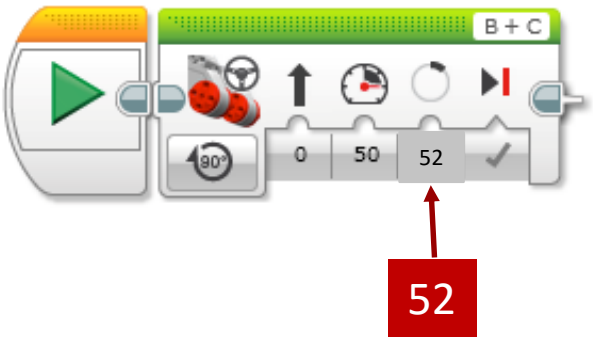
- Much easier to measure distance with a ruler than degrees or rotations.

- 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

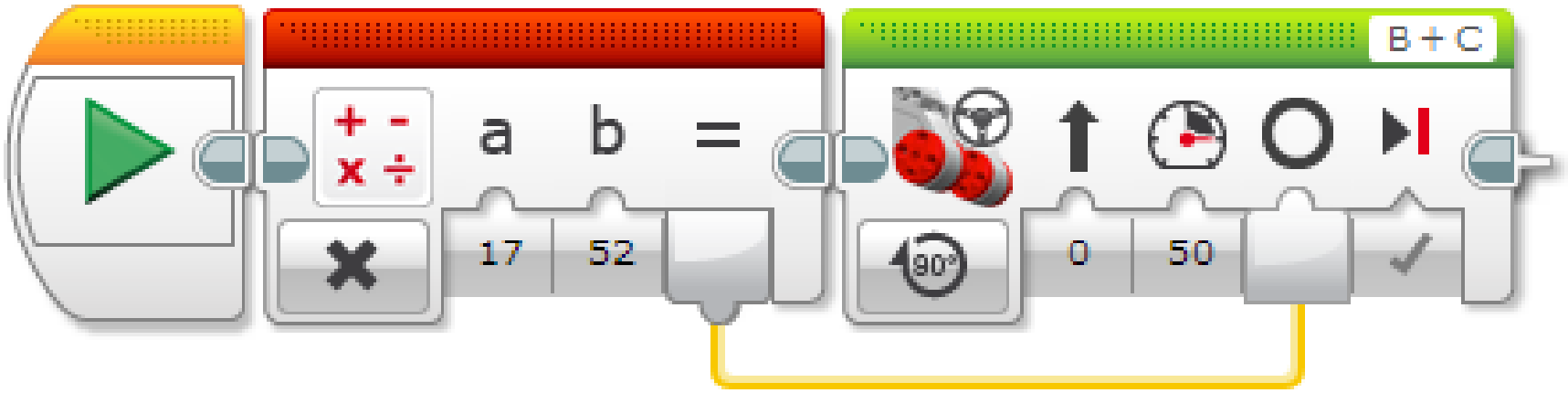


EV3 Base Kit Wheel =
56mm diameter ÷ 25.4 (mm/inch) =
2.2 inch

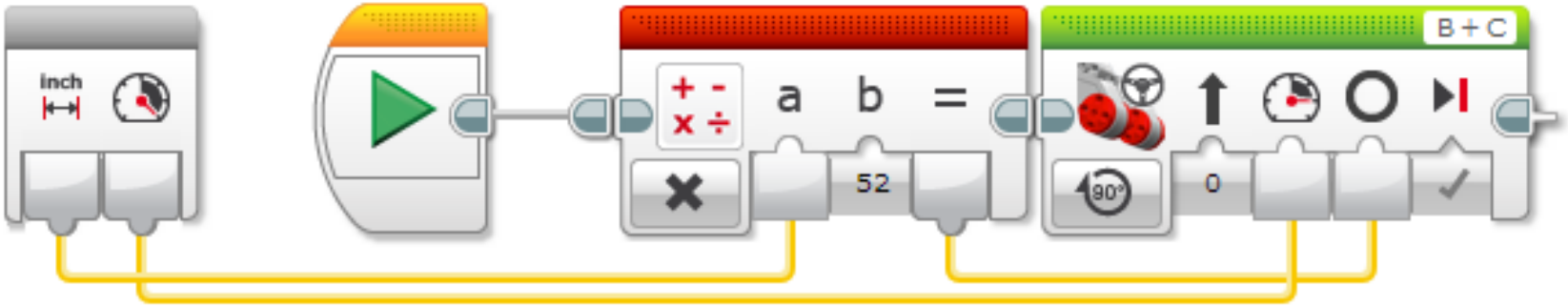
2.2 inches × π = 6.93 inches in each rotation

360 degrees ÷ 6.93 inches = 52 degrees/inch

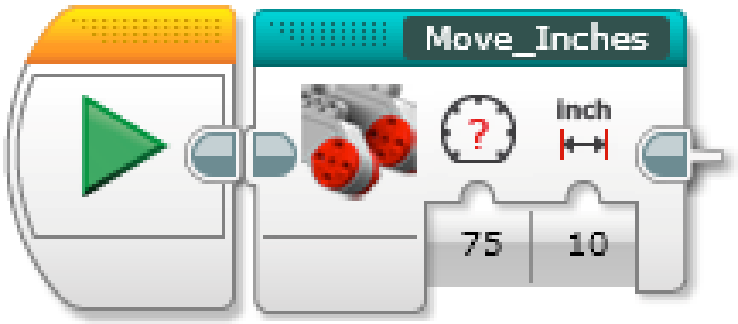
PHASE 2: Math Blocks



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

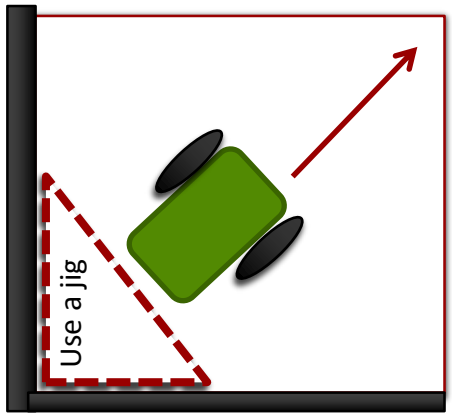
By Droids Robotics

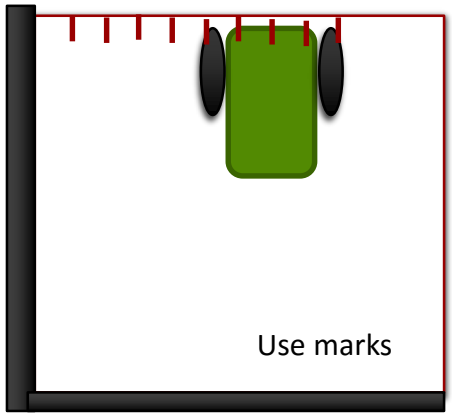
Common Sources of Problems in FLL

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

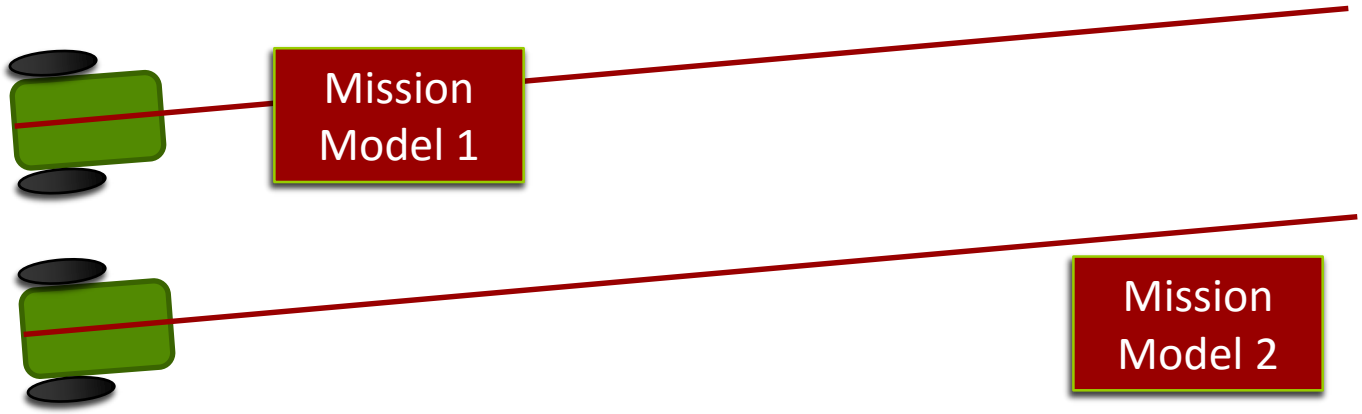
- FLL teams need to figure out where to start in base
  - **Jigs:** a LEGO ruler/wall that your robot can align against them in base
  - **Same start each time:** pick one spot and start there no matter what the mission for easy starts
  - **Tick marks:** Use the inch marks to pick a starting spot for 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

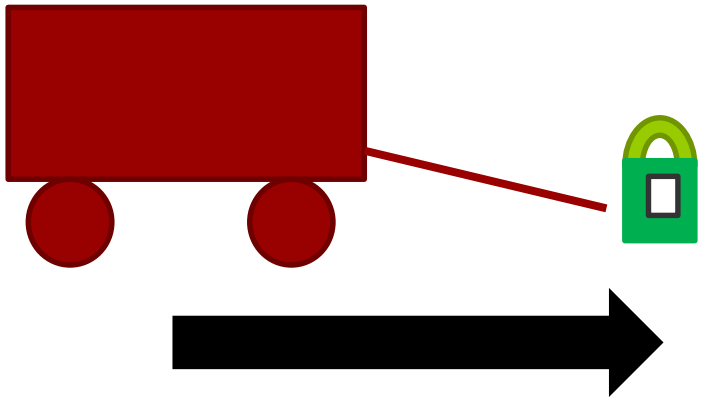
Parallel Beams for Squaring on Lines

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

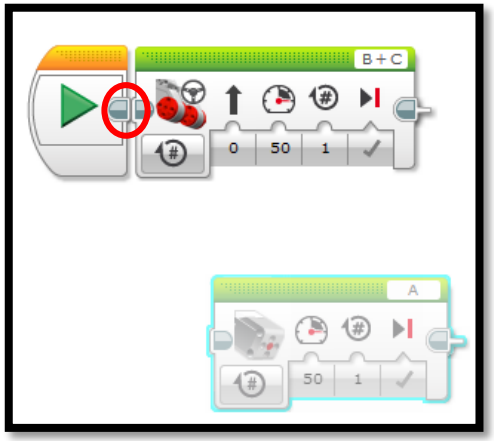


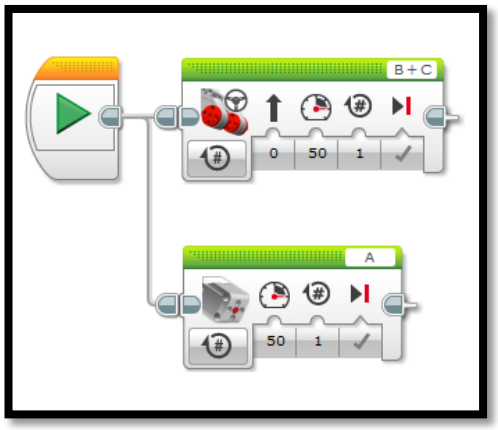
Robot lifting up hoops and driving forward.

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?

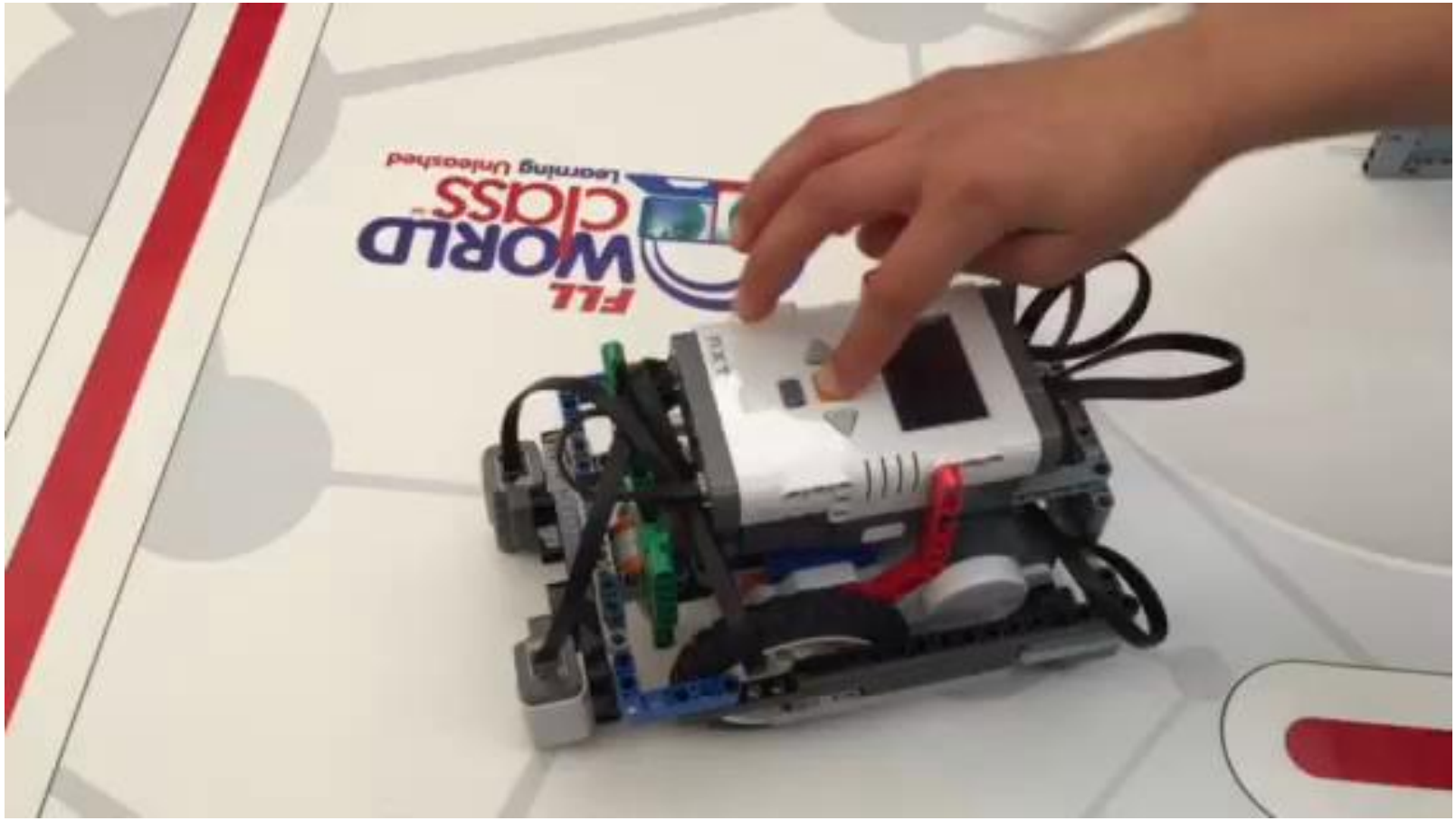
- To learn about Parallel Beams and their limitations/uses, please visit the Intermediate EV3Lessons.com on Parallel Beams.

ADVANCED EV3 PROGRAMMING LESSON

Squaring or Aligning on a Line

By Droids Robotics

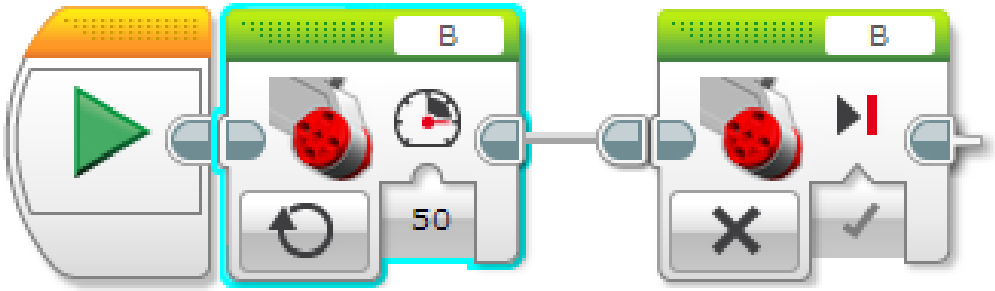
What is Squaring?



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

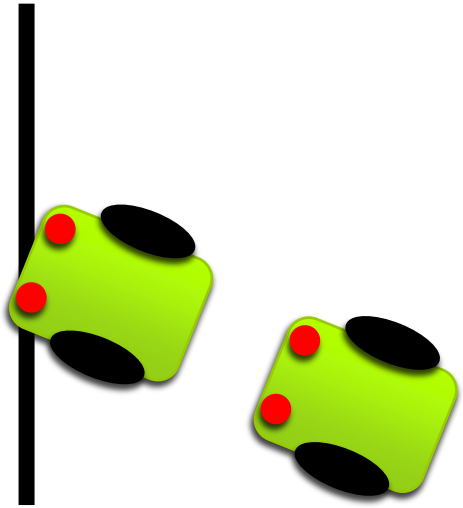
Challenge

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

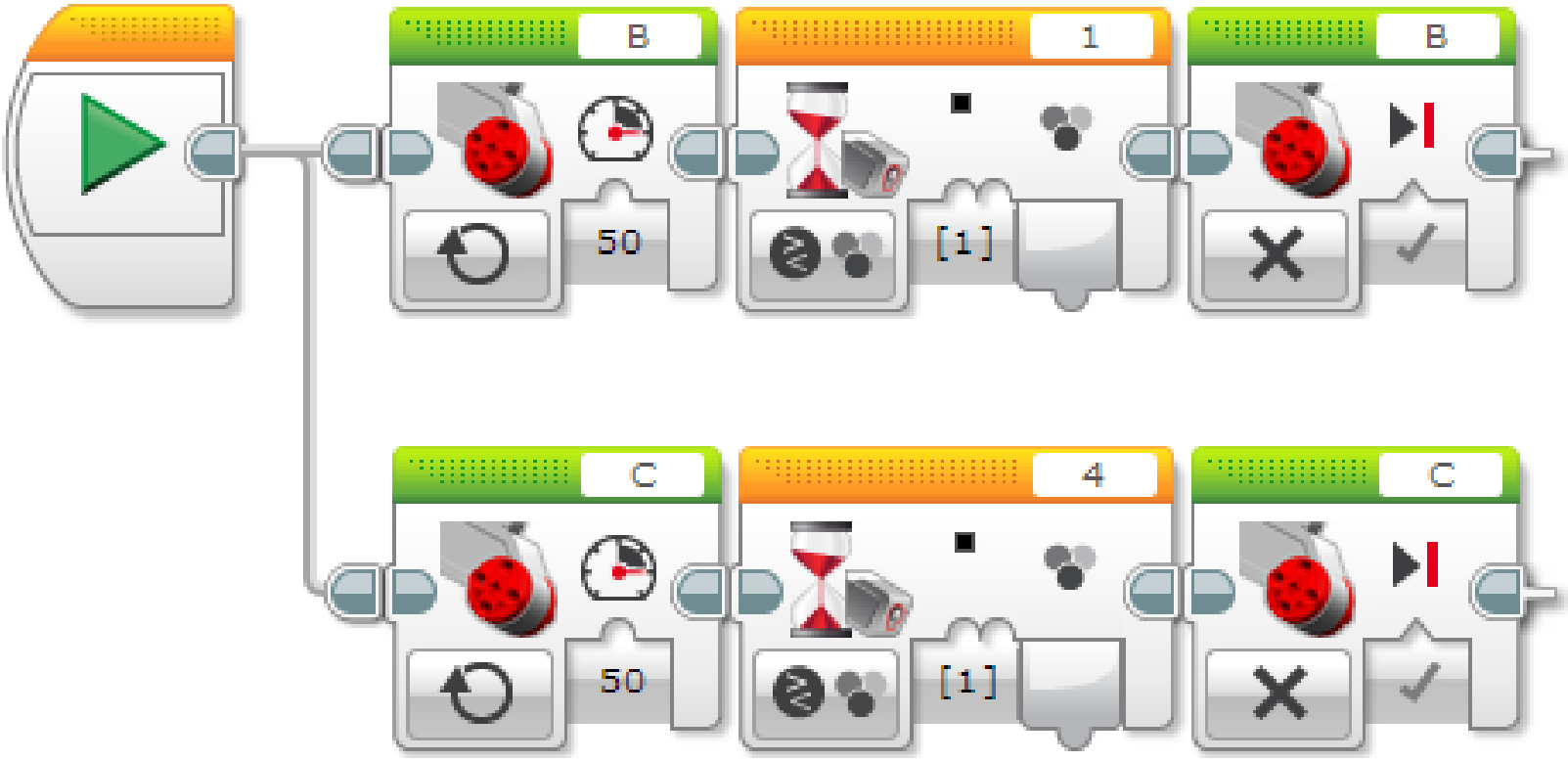
Pseudocode:

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

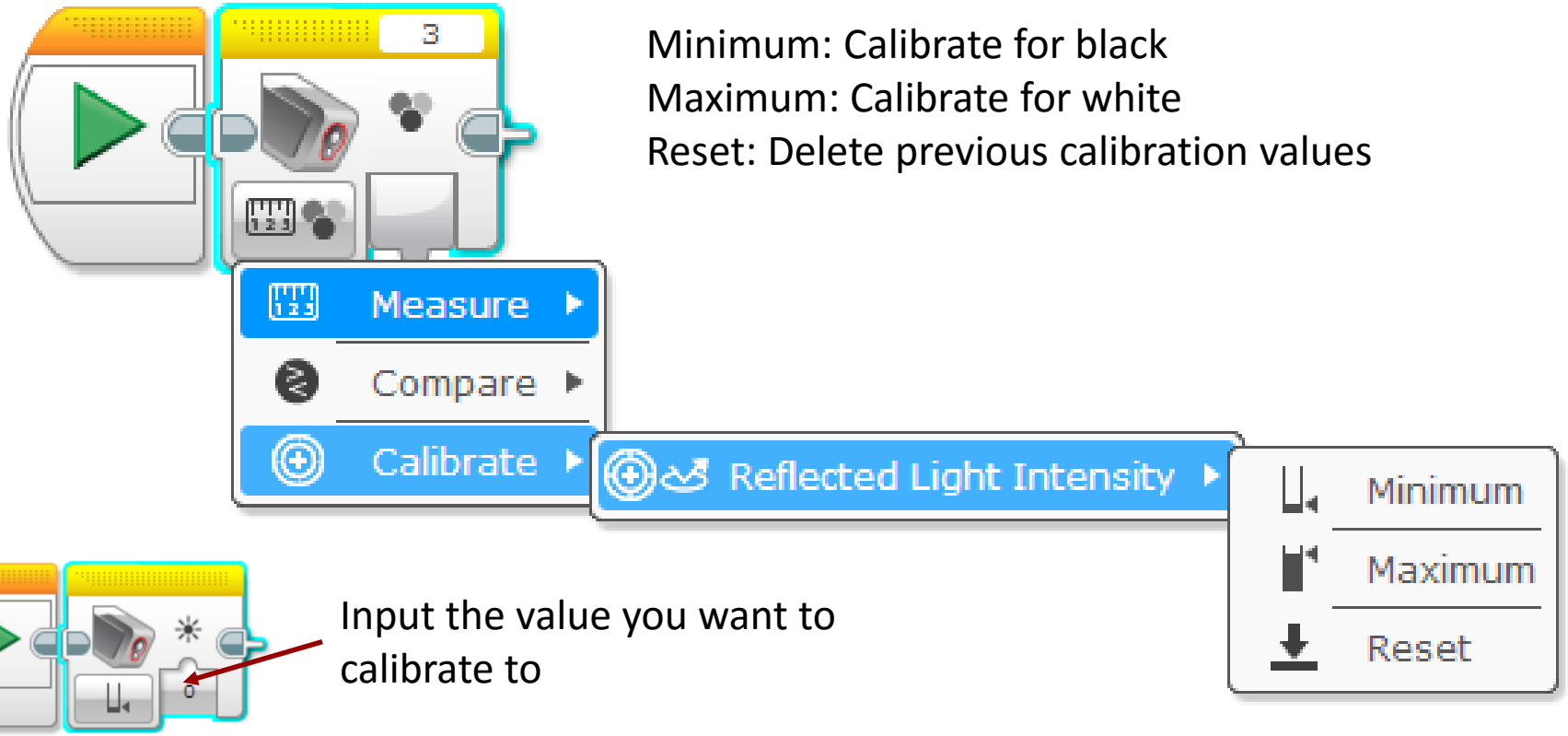
Calibrating Color Sensors for better line followers

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

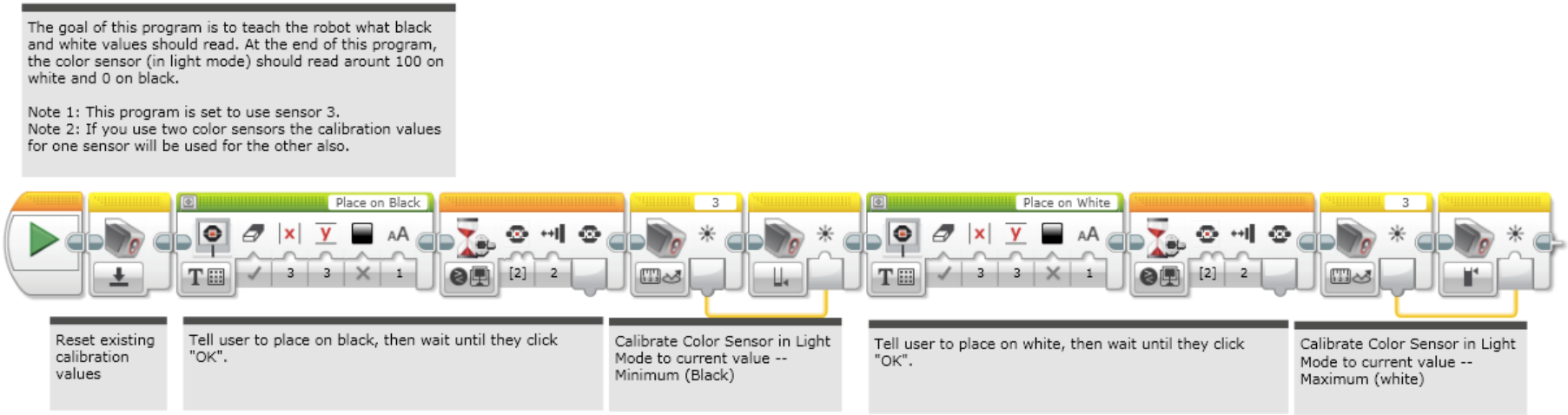
Steps/Pseudocode for Calibration

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

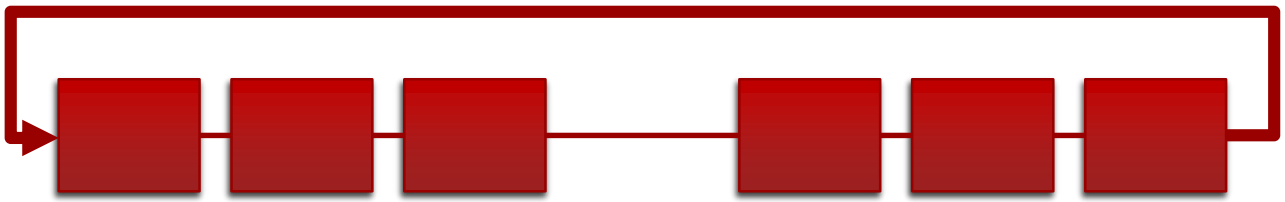
- On our team, we discuss “proportional” as a game.
- Blindfold one teammate. He or She has to get across the room as quickly as they can and stop exactly on a line drawn on the ground (use masking tape to draw a line on the floor).
- The rest of the team has to give the commands.
- When your teammate 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.

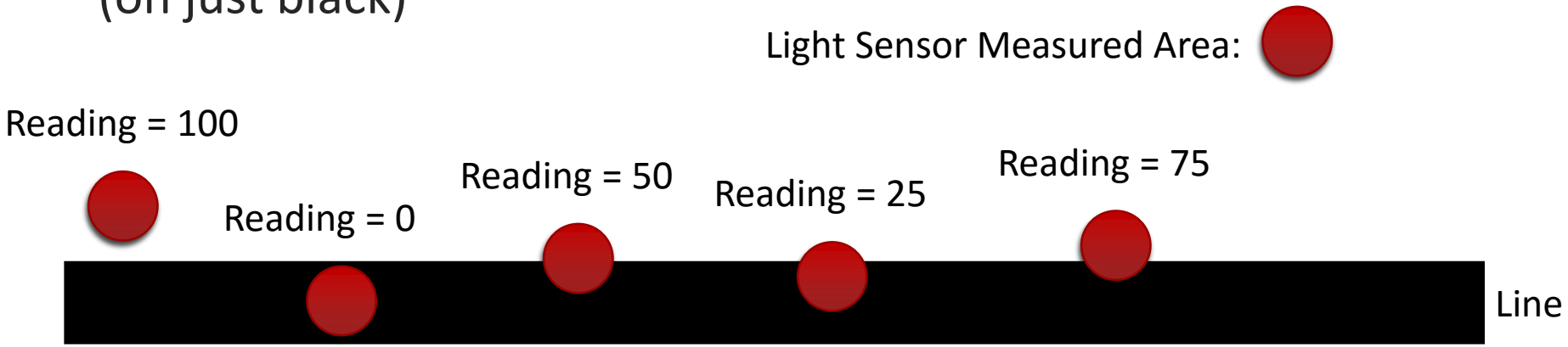


Compute Error

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:

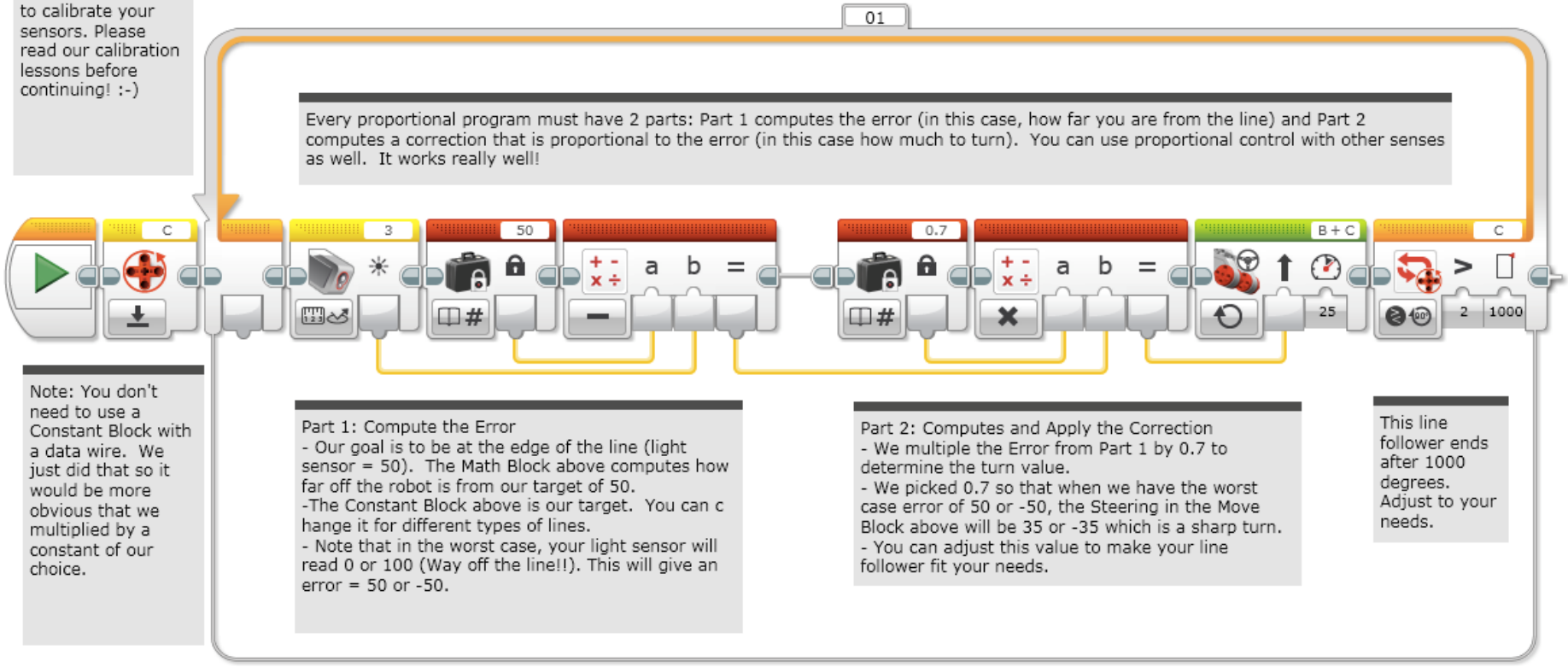
1. Reset the Rotation sensor (Only required for line following for a total distance)
2. Compute the error = Distance from line = (Light sensor reading – Target Reading)
3. Scale the error to determine a correction amount. Adjust your scaling factor to make you robot follow the line more smoothly.
4.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