Turning Tutorial – Update # 1

Ends:

- Improve Student Understanding of Circular Relationships.
- Enable Students to logically develop an approach to converting a desired robot heading change into software that accurately implements the change

Means:

- Demonstrate math to support student comprehension of the relationships between the two major circles associated with a simple Lego NXT robot
- Provide Students sufficient background to develop their own turning programs.
- Provide example programs
- Culminate Student comprehension by programming the robots to move in any regular polygon clockwise or counterclockwise as directed by the instructor.

Tools:

- Acrobat file titled “Making Precise Turns - Understanding the Relationships between the Robot and its Wheels” which includes:
  - attached MS Office and Open Office file of same presentation
  - attached MS Office and Open Office file of spreadsheet implementing the relationships
  - RoboLab screenshots of straight line and turning programs.
  - RoboLab screenshot of a regular polygon shape program using subroutines developed from the straight line and turn program.
  - NXT-G screenshots of straight line and turning programs
  - Attached MS Office and Open Office file of NXT-G Presentation

Discussion:

Most robot competitions I've participated in seem to require the robots to be driven to a specific location on the robot course, do something there, and then return to a declared home base for the next assignment.

After participating with four different teams and three competitions, this seems the one place were a strong mentoring hand can have a good influence, laying the groundwork for other approaches to controlling robot movement. And for competitions that can rely on odometry to get the robot from one place to another, getting the students through this early step, should enable them to accomplish more with less direct mentor intervention.

Regardless of more sophisticated methods to determine the robot's location relative to another place on the board, these seem to be necessary baseline tools to get the robot from one place to another.

This update includes the previous set of RoboLab screenshots, and an additonal presentation using NXT-G. If there's interest and time, I will explore developing parallel LabView screenshots.

The robot depicted is based on a slightly modified Domabotics model.

Comments and criticism welcome.

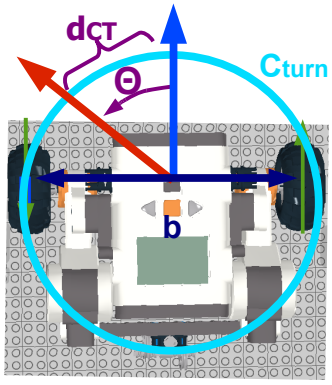
All my best,
Craig Shelden
craig@sheldenrobotics.com

Sheldon Robotics

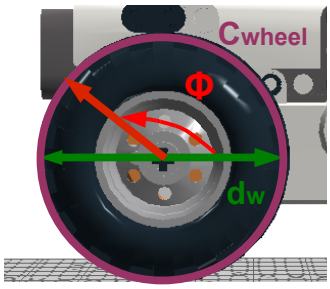
One Page Handout

Making Precise Turns:
Understanding the Relationships between the Robot and its Wheels

Top View



Side View



Terms Defined

b	**Robot Wheelbase**
Cturn	**Robot Circumference of turn**
	Cturn = b x π
Θ	**Desired Turn Angle**
dCT	**Distance along Cturn**
dw	**Wheel Diameter**
Cwheel	**Wheel Circumference**
	Cwheel = dw x π
Φ	**Motor Rotation**
360°	degrees around a circle

We want the robot to turn a set number of degrees (**Θ**) by turning the power wheels in opposite directions. **Θ** can also be expressed in terms of its fractional value of the whole circumference and it is also equal to the distance along the circumference we want to turn the robot.

(**Θ** /360) = (**dCT** / **Cturn**)

And we can solve this for the Distance along the circumference to turn the wheels.

dCT = (**Θ** /360°) x (**Cturn**)

We want to be able to program this in terms of wheel rotations or degrees.

Recognize that the distance along the Robot's Turning Circumference (**dCT**) is the same distance the robot's wheels must turn along the Circumference of each Wheel (**Cwheel**) to make the turn.

This leads to the following relationship:

Number of Rotations to Program = (**dCT**) / (**Cwheel**) in NXT-G

Number of Degrees to Program = (360°) x (**dCT**) / (**Cwheel**) in NXT-G or RoboLab

And if we substitute the terms that make **dCT** into the above equations, they simplify rather nicely.

Number of Rotations to Program = (**Cturn**) / (**Cwheel**) x (**Θ**) / (360°) in NXT-G

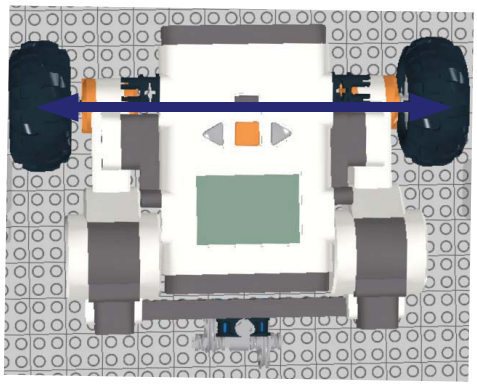
Number of Degrees to Program = (**Cturn**) / (**Cwheel**) x (**Θ**) in NXT-G or RoboLab

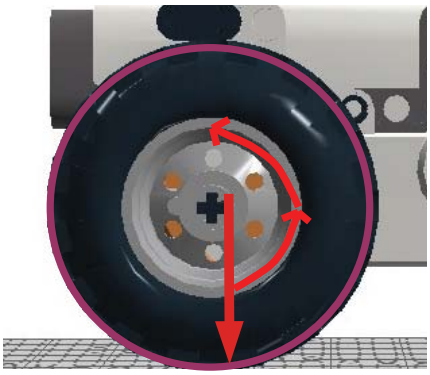
Detailed Turning Tutorial Slides

Making Precise Turns

Understanding the Relationships between the

Robot and its Wheels



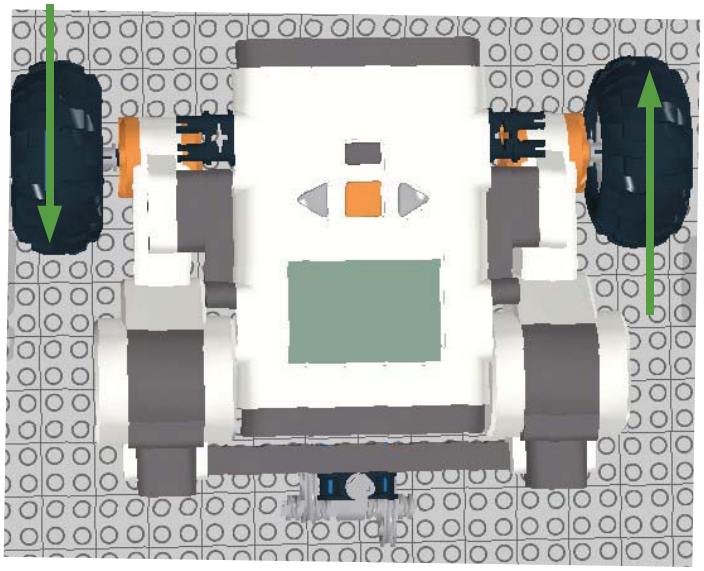


Turn Concept

We're going to turn the robot by driving one wheel backward and one wheel forward.

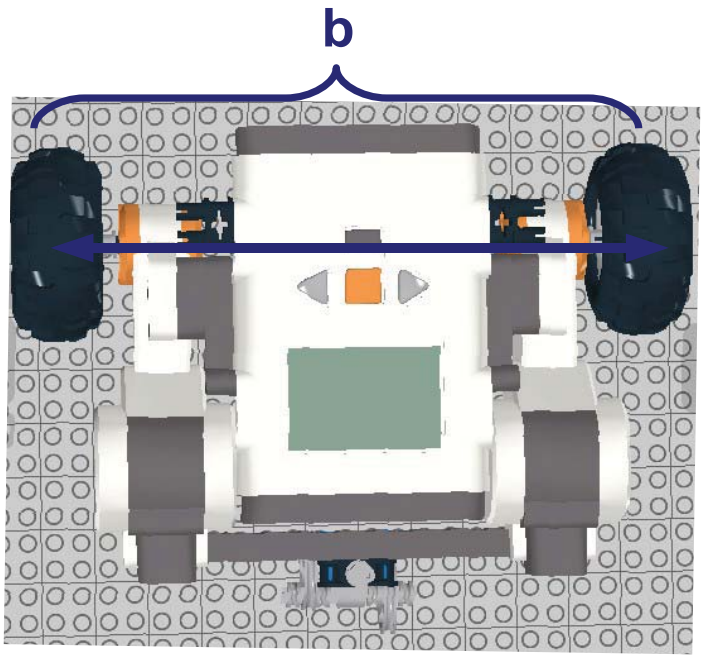
Telling the robot to turn each motor “some amount” forces the wheel to travel along **C_{turn}**.

The distance along **C_{turn}** – when moved ahead on one side and backwards on the other, turns the robot.



Wheelbase

The distance between Robot wheel centers is called the **Wheelbase** (**b**)



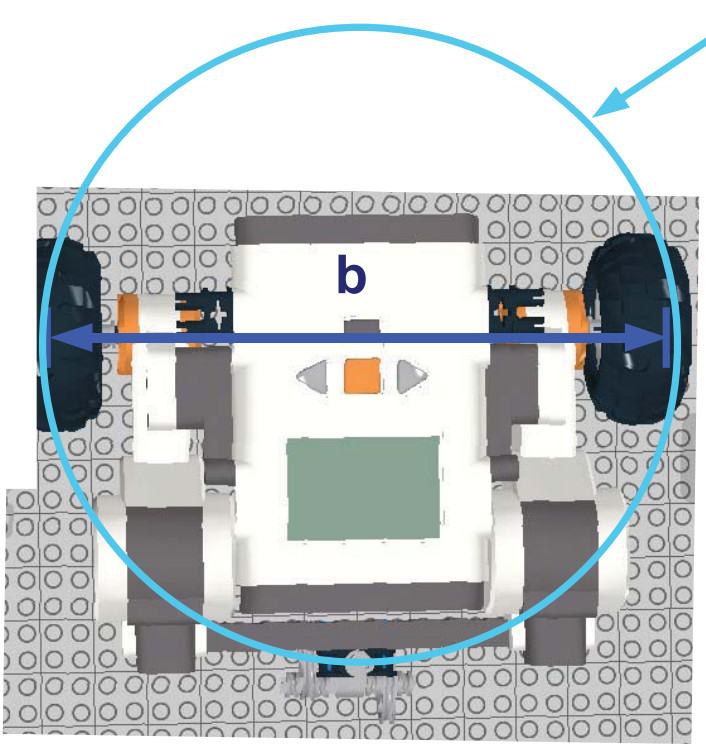
Turning Circle or Turning Circumference

The **Wheelbase** is also the **Diameter** of the Robot's **Turning Circumference** (**Cturn**)

The turning circumference defines how tight a turn the robot can make.

The length of the Turning Circumference is defined as

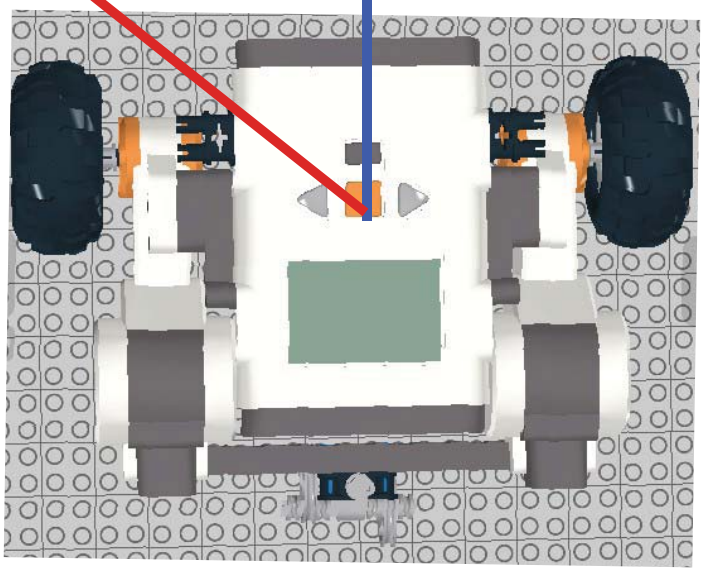
$C_{turn} = b \times \pi$



But we need to know how far to turn it: Robot Heading

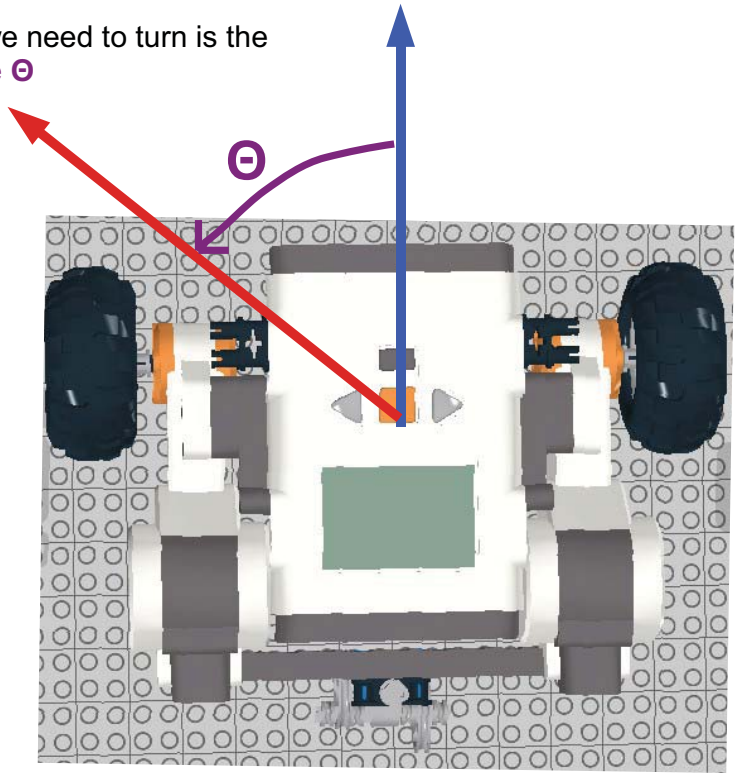
Desired direction we want the robot to turn to is called its **New HEADING**

Current direction the robot is pointing is called its **HEADING**

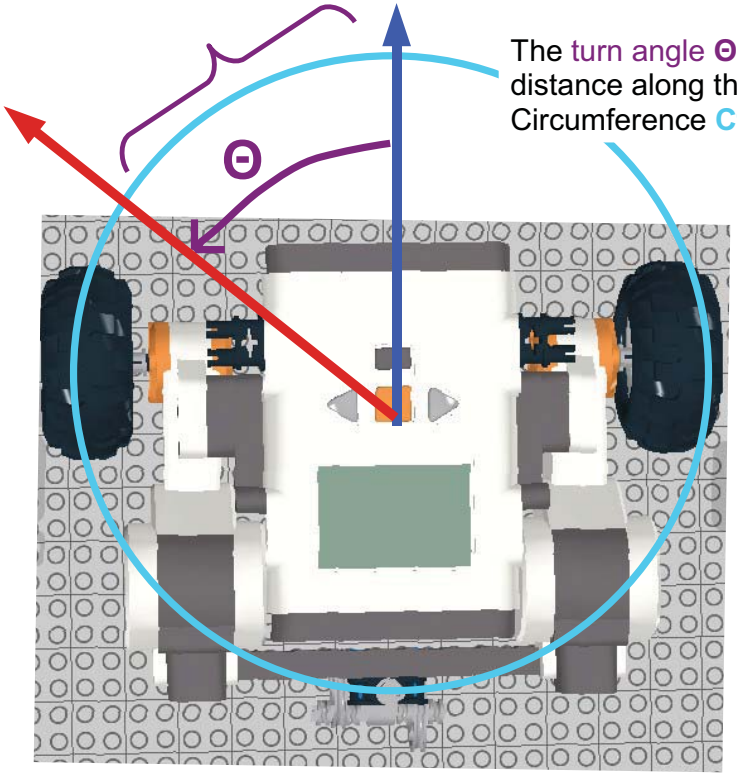


Now for Turn Angle (Θ) Robot Heading

How far we need to turn is the turn angle **Θ**

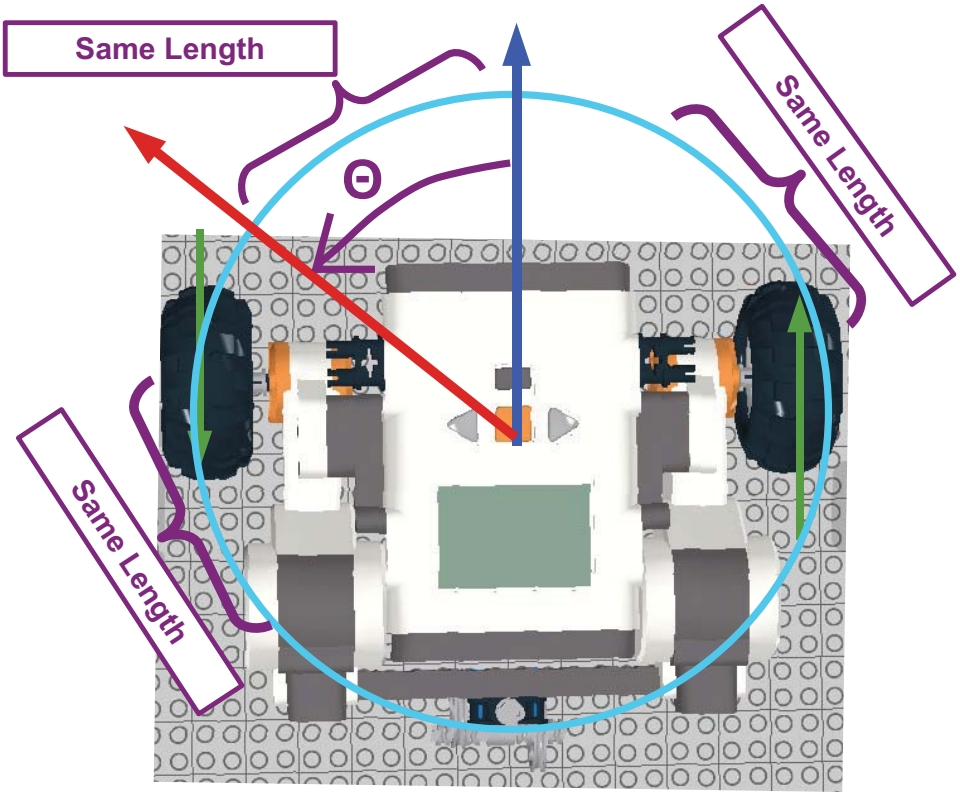


Now for Turn Angle (Θ)
Defining Robot Heading



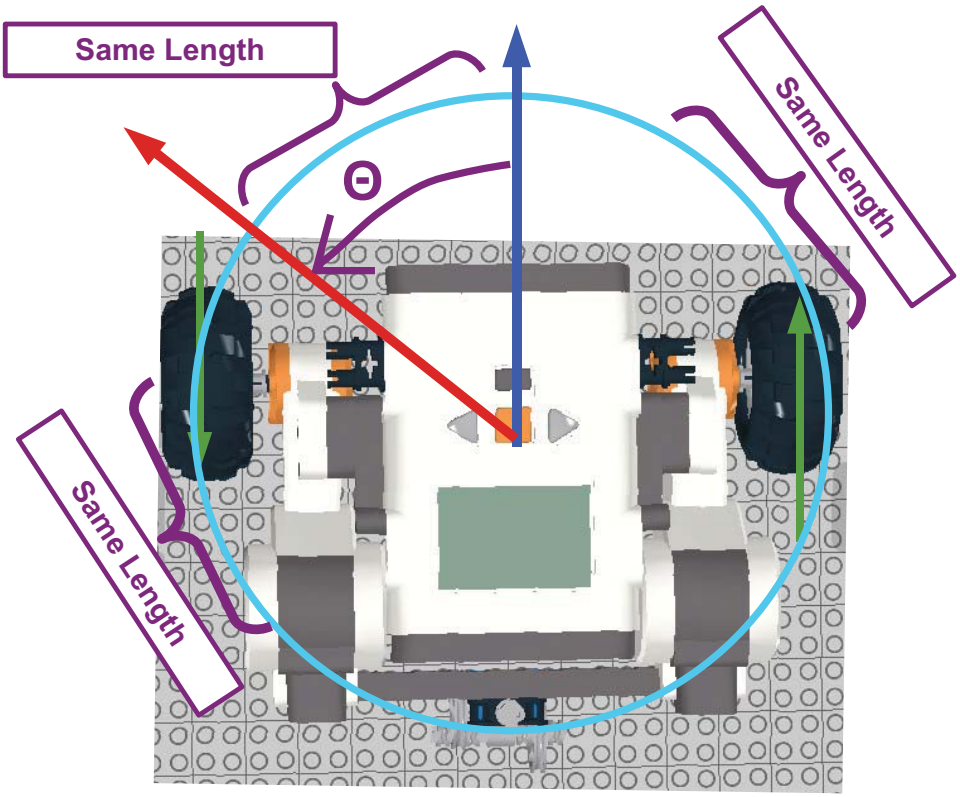
The turn angle **Θ** corresponds to a distance along the Turning Circumference **Cturn**

Turn Concept



All the purple brackets { are the **same length.**

Giving "Same Length" a Name → dCT



All the purple brackets { are the **same length.**

Figuring Out How Far to Turn

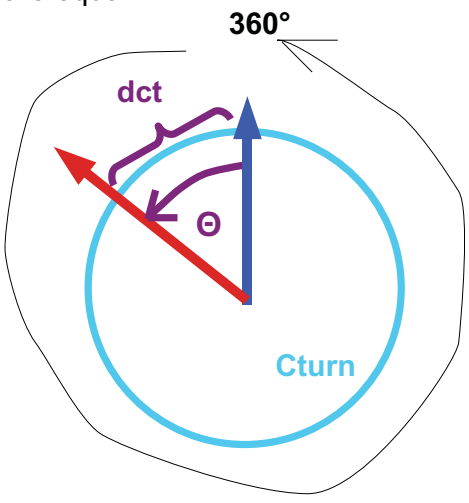
dCT or "**Distance along Cturn**" can be expressed as a fraction of the Robot's **Turning Circumference** (**Cturn**)

And

The degrees of heading change we want to make (**Θ**) can be expressed as a fraction of the # of degrees in a circle (360°).

These two fractions both describe the same thing – and are equal.

$$\frac{dCT}{Cturn} = \frac{\Theta}{360°}$$



Figuring Out How Far to Turn

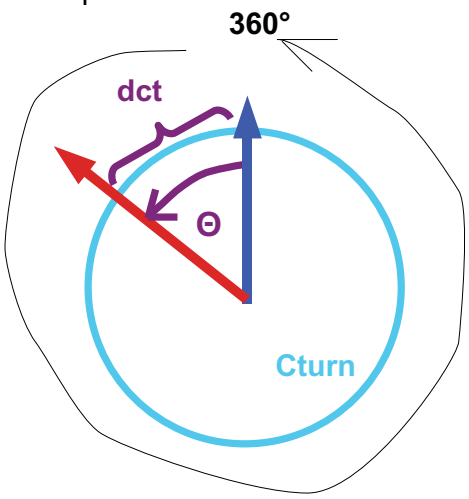
dCT or "**Distance along Cturn**" can be expressed as a fraction of the Robot's **Turning Circumference** (**Cturn**)

And

The degrees of heading change we want to make (**Θ**) can be expressed as a fraction of the # of degrees in a circle (360°).

These two fractions both describe the same thing – and are equal.

$$\frac{dCT}{Cturn} = \frac{\Theta}{360°}$$



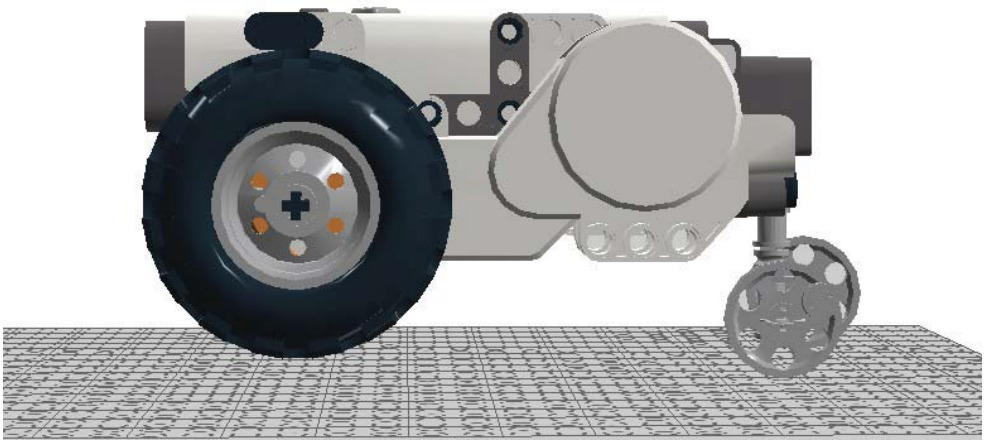
And we can solve for **dCT**

$$dCT = Cturn \times \frac{\Theta}{360°}$$

But we need to tell the motors how much to turn
So there are a few more things to look at

Remember, the motors don't know how far they've traveled, only the degrees or rotations they've made.

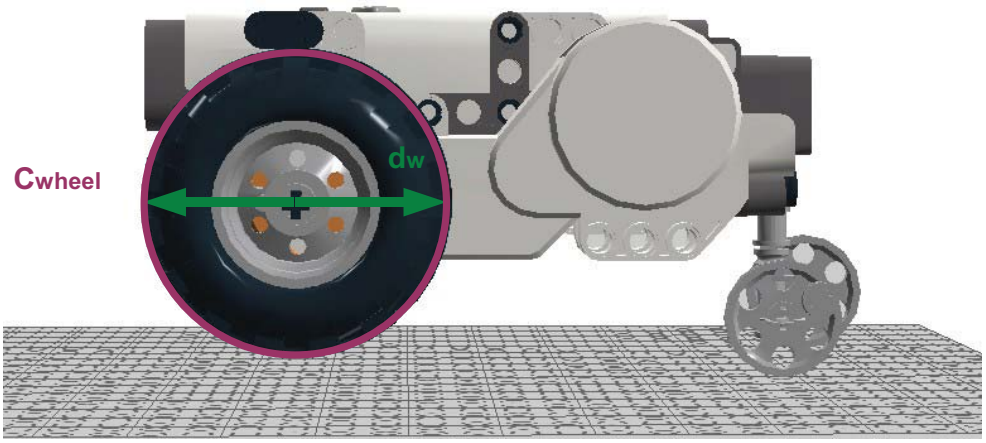
How far they travel – "some amount" – is up to the Programmer.



Motor Rotation – Just a few more terms

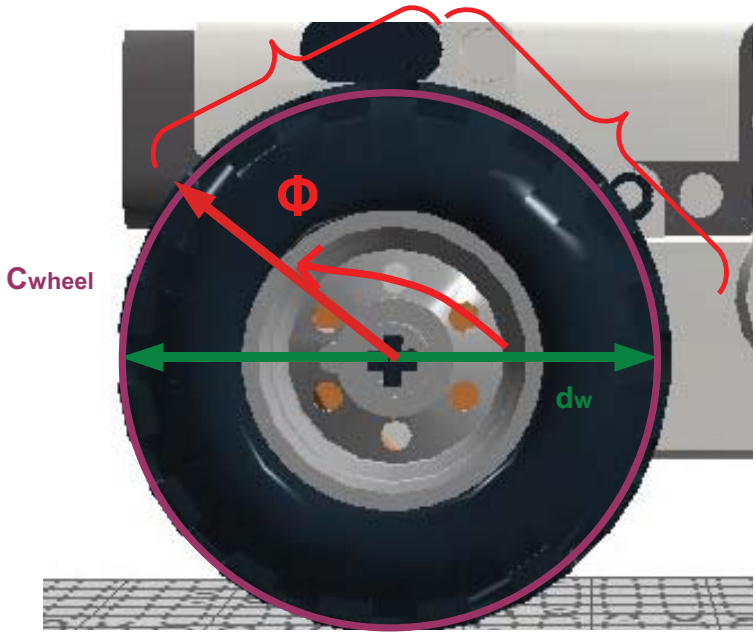
The **Diameter of the Wheel (dw)** and **Wheel Circumference (Cwheel)** determine how far the wheels of the robot move for a given number of degrees.

$$C_{wheel} = d_w \times \pi$$



Motor Rotation – Finally something we can Program!

We tell the motors how much to turn each wheel using either degrees (RoboLab & NXT-G) or Rotations (NXT-G). **Wheel Turn Angle (Φ)** is how far we tell the motor to turn the wheel.



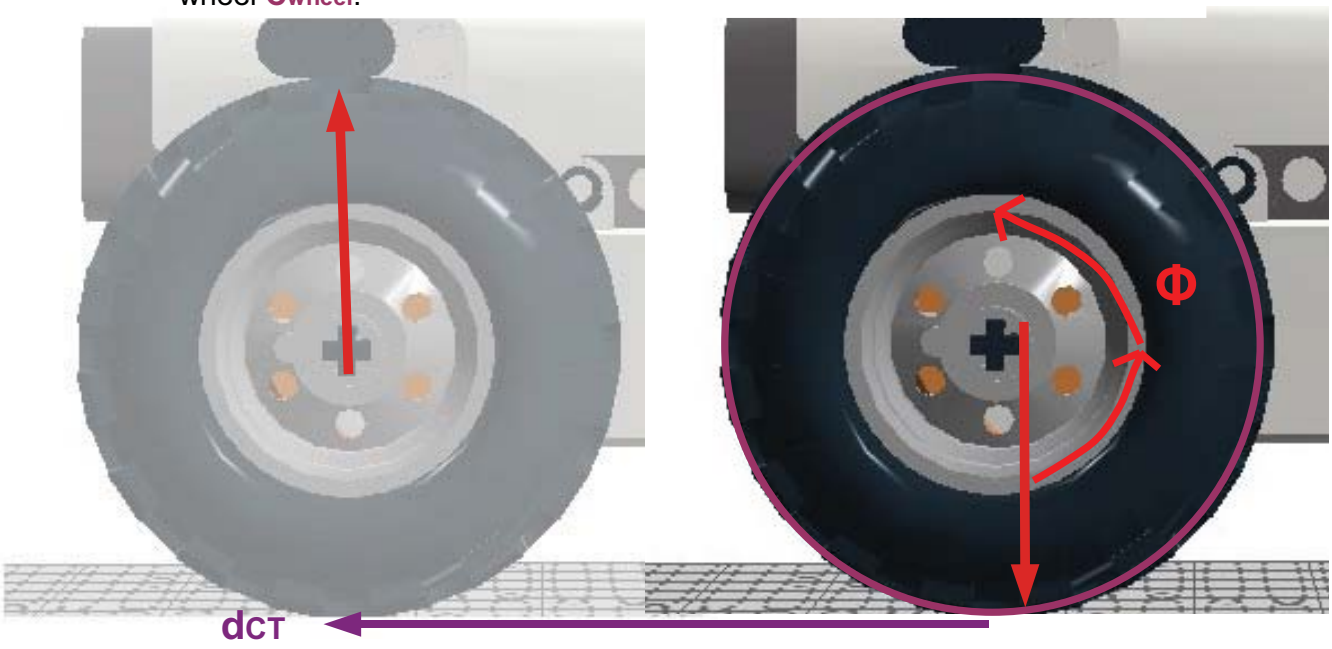
Telling the robot to turn the motor by **Φ** forces the wheel to travel along **Cturn**.

The distance along **Cturn** – when moved ahead on one side and backwards on the other, turns the robot.

Motor Rotation – Finally something we can Program!

Let's look at how we can express d_{CT} in terms of the wheels and the Wheels' turn angles.

In the picture below we're moving the robot from right to left by rotating the wheel an angle Φ that corresponds to d_{CT} along the circumference of the wheel C_{wheel}.

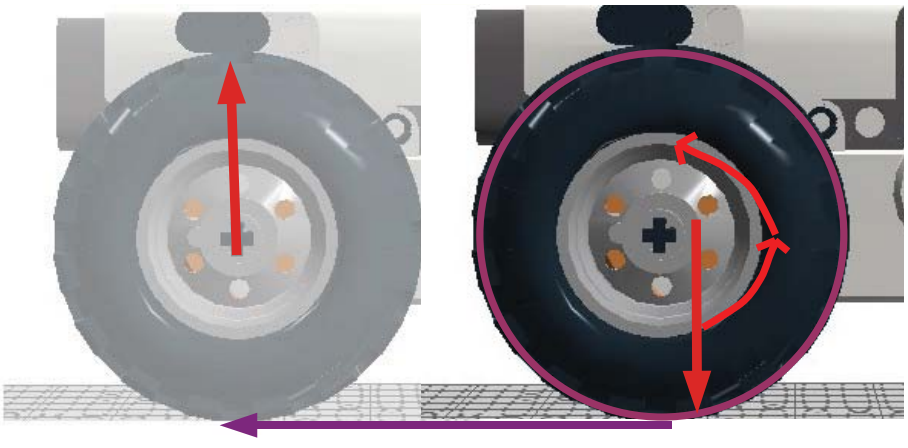


Motor Rotation – Finally something we can Program!

And now we can see that d_{CT} is a fraction of C_{wheel} as well as a fraction of C_{turn}.

And how far to tell the program to move the wheel:

$$\textbf{\# of Wheel Motor Rotations} = \frac{d_{CT}}{C_{wheel}}$$ For NXT-G



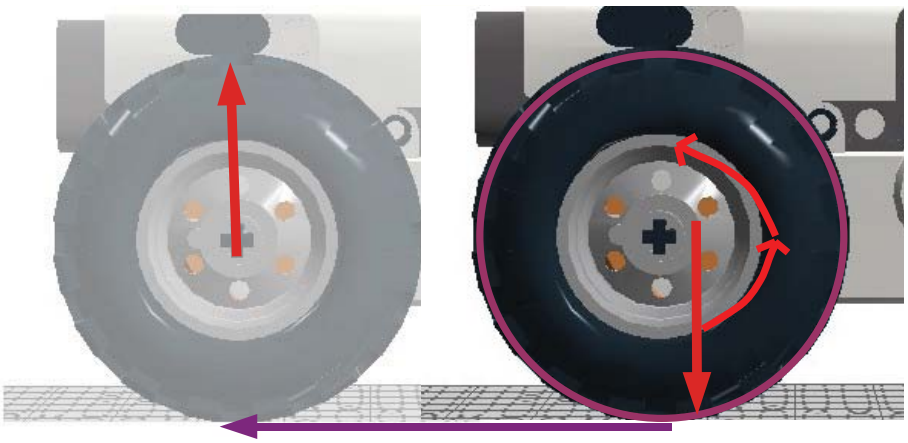
Motor Rotation – Finally something we can Program!

And now we can see that d_{CT} is a fraction of C_{wheel} as well as a fraction of C_{turn}.

And how far to tell the program to move the wheel:

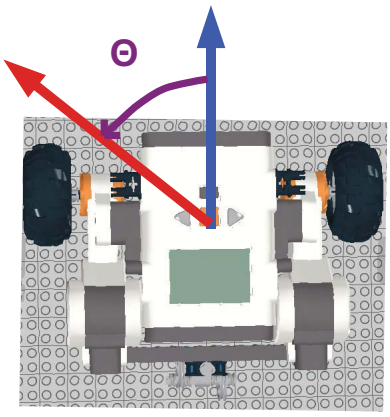
$$\text{\# of Wheel Motor Degrees} = 360° \times \frac{d_{CT}}{C_{wheel}}$$

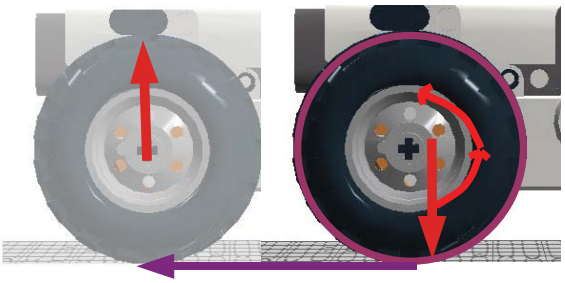
For NXT-G and Robolab



Motor Rotation – Don't be Surprised

For large Turn Angles Θ or small wheels, d_{CT} is sometimes a fraction that is bigger than one.



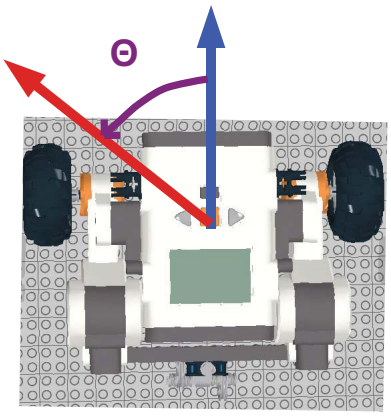


Summary

Based on our mission, we know how many degrees we need to turn the robot, or our desired turn angle **Θ.**

And Θ corresponds to a fraction (d_{CT}) of our robot's turning circumference (**Cturn**):

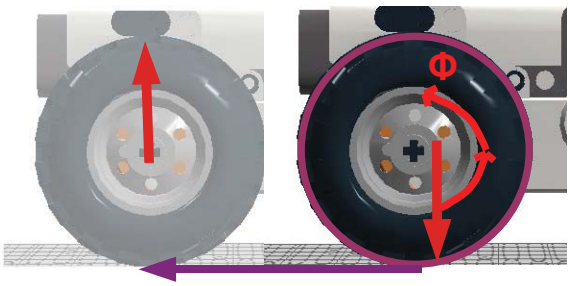
$$d_{CT} = \text{Cturn} \times \frac{\Theta}{360°}$$



And d_{CT} is **also** a fraction of C_{wheel}

$$\text{\# of Wheel Motor Rotations} = \frac{d_{CT}}{C_{wheel}}$$

$$\text{\# of Wheel Motor Degrees} = 360° \times \frac{d_{CT}}{C_{wheel}}$$



Summary – 2 Combining terms

Based on our mission, we know how many degrees we need to turn the robot, or our desired turn angle **Θ.**

And Θ corresponds to a fraction (d_{CT}) of our robot's turning circumference (**Cturn**):

And d_{CT} is **also** a fraction of C_{wheel}

Substituting d_{CT}'s factors into the # of Rotations or Degrees has some interesting results.

$$d_{CT} = \text{Cturn} \times \frac{\Theta}{360°}$$

$$\text{\# of Wheel Motor Rotations} = \frac{d_{CT}}{C_{wheel}} = \frac{\text{Cturn}}{C_{wheel}} \times \frac{\Theta}{360°}$$

$$\text{\# of Wheel Motor Degrees} = 360° \times \frac{d_{CT}}{C_{wheel}} = 360° \times \frac{\text{Cturn}}{C_{wheel}} \times \frac{\Theta}{360°}$$

Summary – 2 Combining terms

Based on our mission, we know how many degrees we need to turn the robot, or our desired turn angle **Θ.**

And **Θ** corresponds to a fraction (**d_{CT}**) of our robot's turning circumference (**Cturn**):

And **d_{CT}** is **also** a fraction of **C_{wheel}**

Substituting **d_{CT}**'s factors into the # of Rotations or Degrees has some interesting results that make all the math EASIER.

$$d_{CT} = \text{Cturn} \times \frac{\Theta}{360°}$$

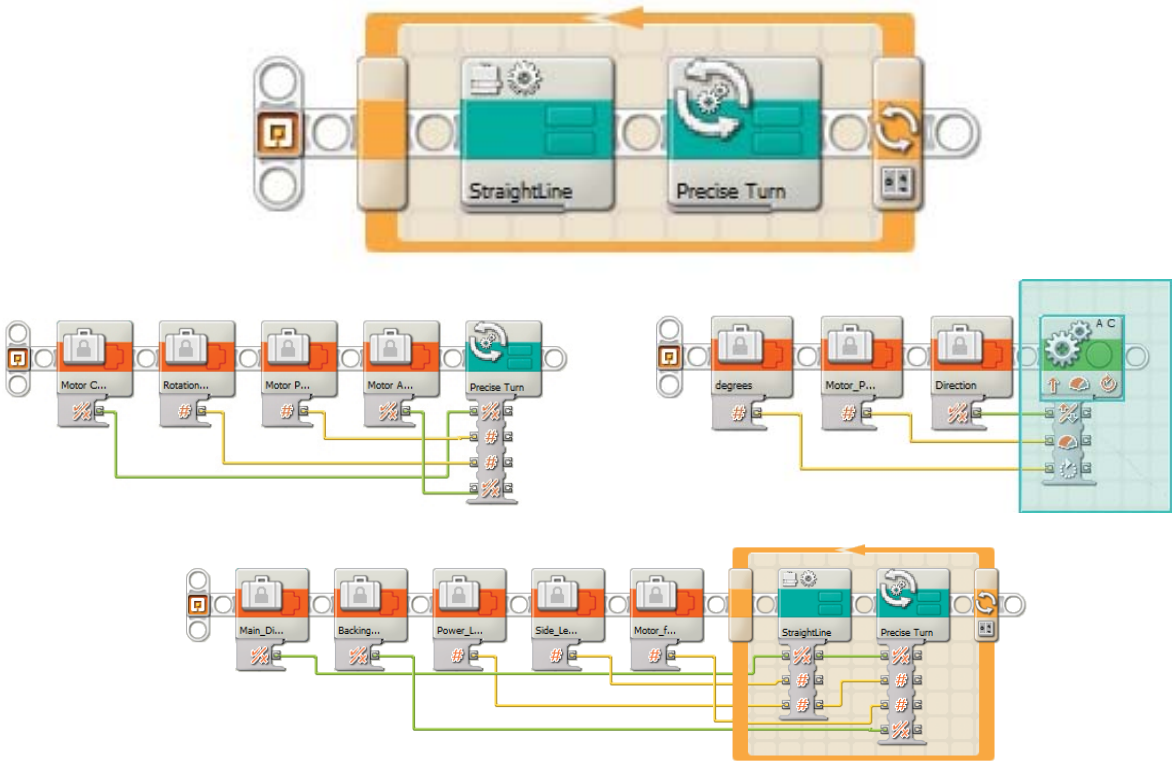
$$\text{\# of Wheel Motor Rotations} = \frac{d_{CT}}{C_{wheel}} = \frac{\text{Cturn}}{C_{wheel}} \times \frac{\Theta}{360°}$$

$$\text{\# of Wheel Motor Degrees} = 360° \times \frac{d_{CT}}{C_{wheel}} = \cancel{360°} \times \frac{\text{Cturn}}{C_{wheel}} \times \frac{\Theta}{\cancel{360°}}$$

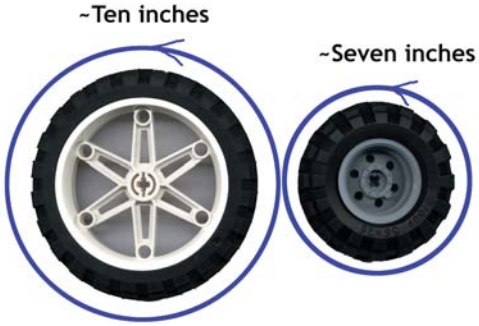
$$\text{\# of Wheel Motor Degrees} = \frac{\text{Cturn}}{C_{wheel}} \times \Theta$$

NXT-G Presentation

Creating Regular Polygon Paths in NXT-G



How Long Are the Sides?



The lengths of the polygon sides are set by the programmer

For these two common Lego wheels, one rotation equals 7 or 10 inches.

It follows therefore that 10 inches = 360° or 7 inches = 360°

Or

$distance_{large\ wheel}$ = (360°/10 inches) = (36°/inch)

$distance_{small\ wheel}$ = (360°/7 inches) = (°51.3/inch)

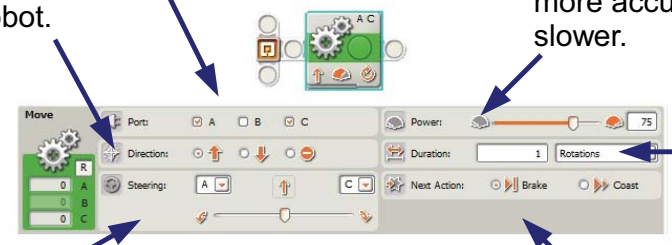
Can use the conversion factor to determine the number of rotations for any distance.

Start with the Basic Move Block

Port → choose the two motor ports for your robot.

Direction → choose whichever corresponds to Forward for your robot.

Power → lower generally provides more accurate control, but is also slower.

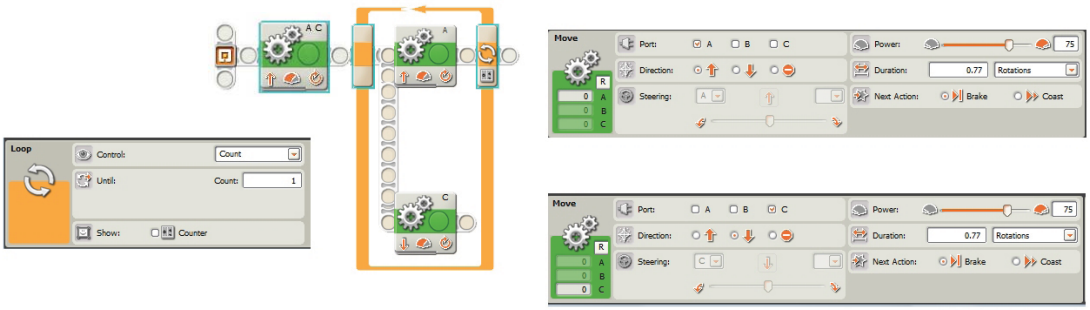


Duration → length of polygon sides

Steering → leave centered.

Braking → keeps the corners sharp

Then Add a Loop and Split to Control the Motors Simultaneously



You don't have to put the A & C motors inside a loop, but it makes the turns stand out visually while working in a lengthy program.

Note that A & C motors are turning in opposite directions.

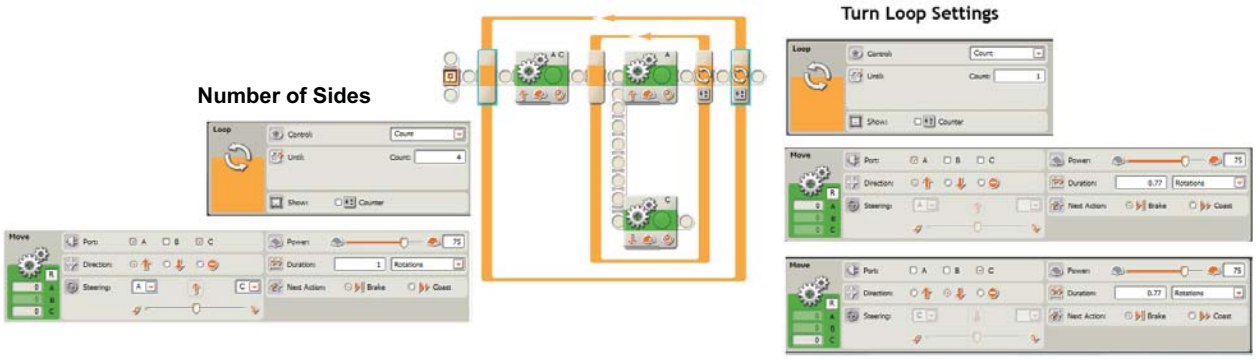
Now we've built the first side and first turn of our polygon.

Hint: Hold down the Shift Key to split the sequence beam...

How Far to Turn Each Wheel?

Value of turn in rotations (or degrees) calculated based on the required change in robot heading for each turn of the polygon.

Then Nest Them in a Loop



The outside loop controls the **number** of polygon sides.

The AC move block controls the **length** of the polygon sides.

The internal loop controls the **inside angle** of each turn.

And we have a basic polygon driving program.

This one will make a square for a robot with a 7 inch wheelbase and the small balloon tires, which are about 7 inches around.

What About Those Subroutines?

This tutorial was supposed to include something about subroutines, wasn't it?

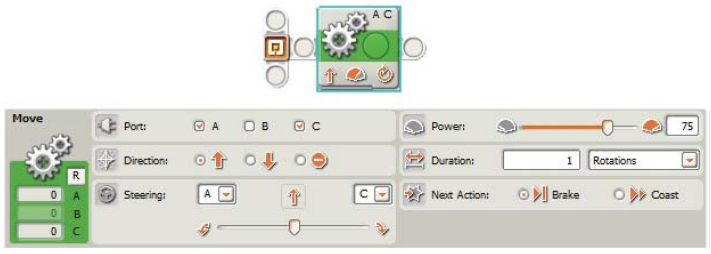
RoboLab allows subroutines to be defined inside the program.

NXT-G and LabView do things different.

- NXT-G uses **MyBlocks**
- LabView uses **SubVIs**

We'll take a look at this same problem using NXT-G MyBlocks

Building the Straight Line MyBlock



Look at the straight line portion of the previous program.

We need to decide what we want to control, and what we want to work the same every time we run the program.

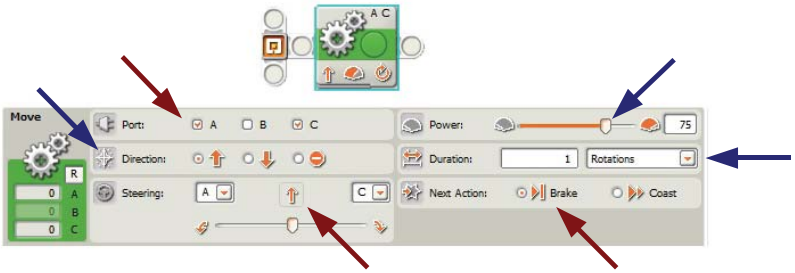
The things we want to control, we need to make sure we can control.

The things we want to leave alone, we can ignore in the next few steps.

But we should figure them out now.

- It's OK if we choose wrong, we can always come back and fix it.

Building the Straight Line MyBlock



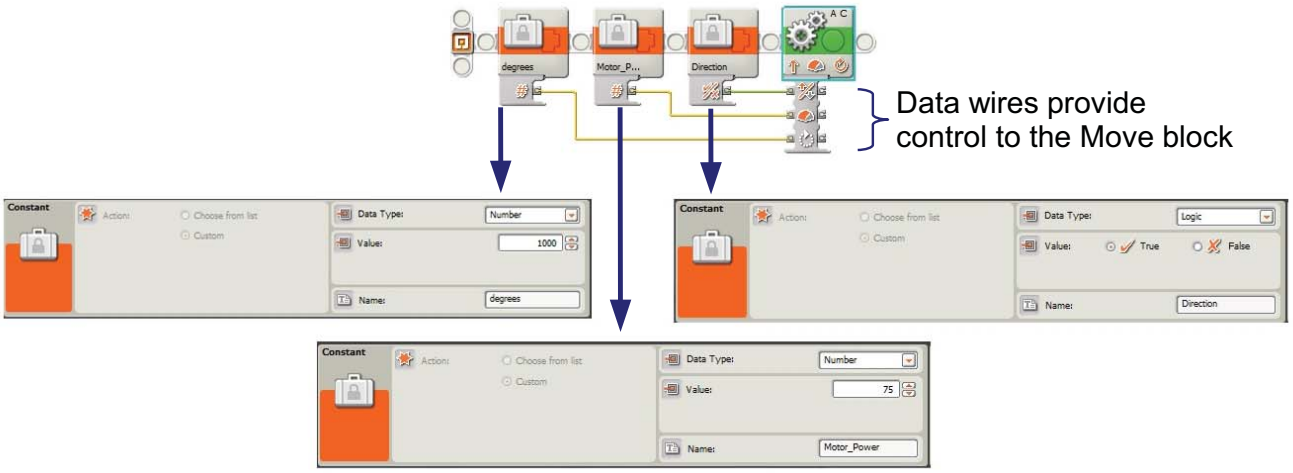
For this example, we want to control

- **Direction**
- **Power**
- **Duration**

And we will leave alone:

- **Motors on ports A & C**
- **Steering**
- **Braking**

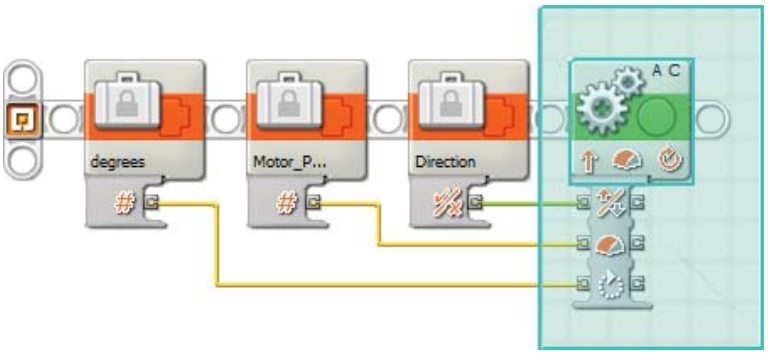
Building the Straight Line MyBlock

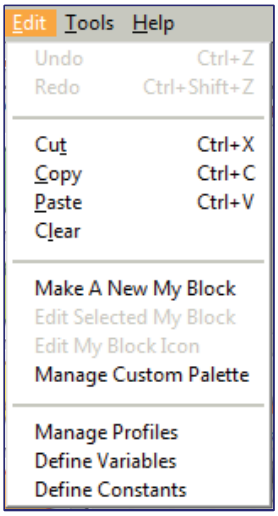


Use Constant blocks from the Data Palette to set acceptable values for the motor block.

Use data wires to connect the constants to the motor block.

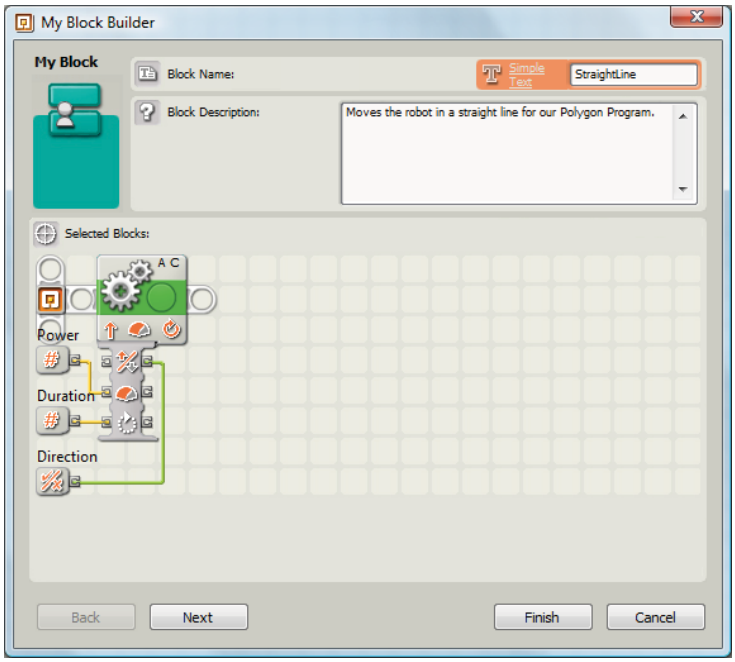
Building the Straight Line MyBlock





Highlight the move block, and from the **`Edit`** menu, choose, **`Make a New My Block`**

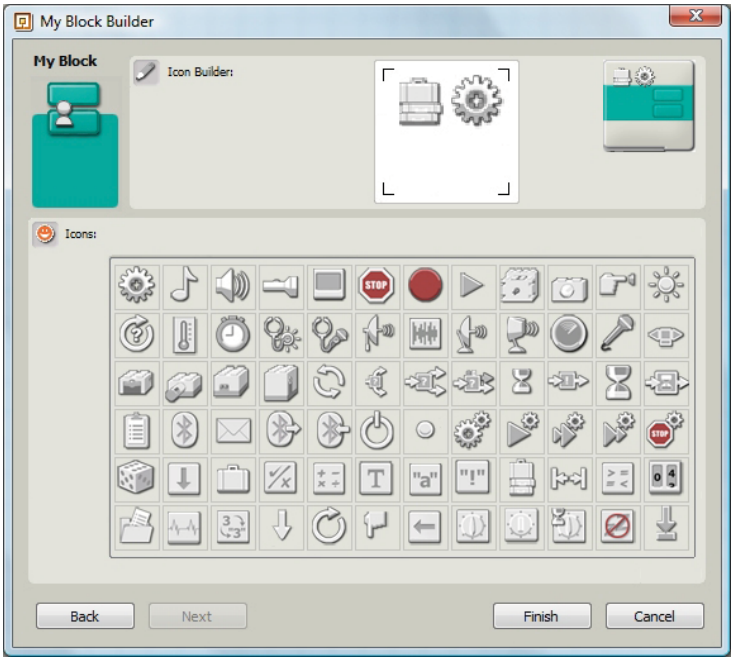
Building the Straight Line MyBlock



Selecting **`Make a New My Block`** opens up this window.

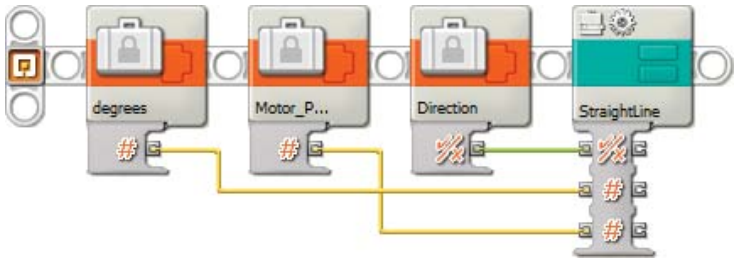
Notice how our inputs are now visible on the left side of the building window.

Building the Straight Line MyBlock



Decorate the new icon so it will make sense.

Building the Straight Line MyBlock



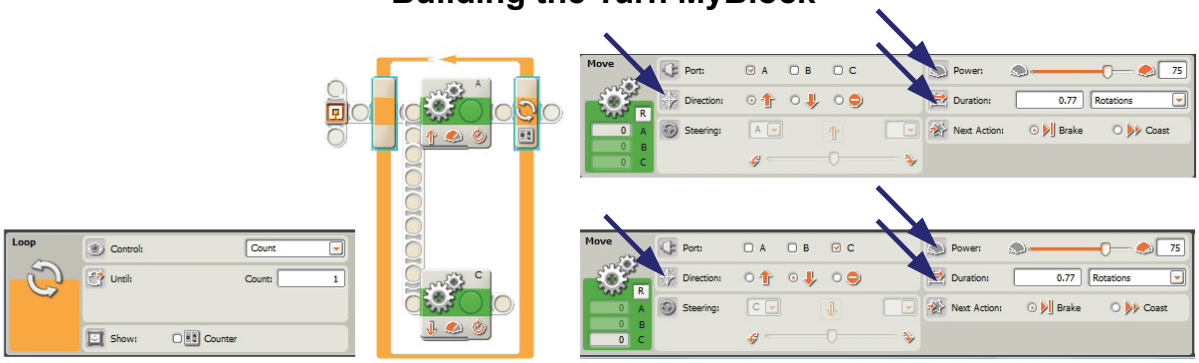
When completed, the NXT-G window will look like this.

Seems like a lot of work...

This seems like an awful lot of work to trade one Move block for one MyBlock.

To see what we gain, let's make a Turn MyBlock and put the two together.

Building the Turn MyBlock



Look at the turn portion of the previous program.

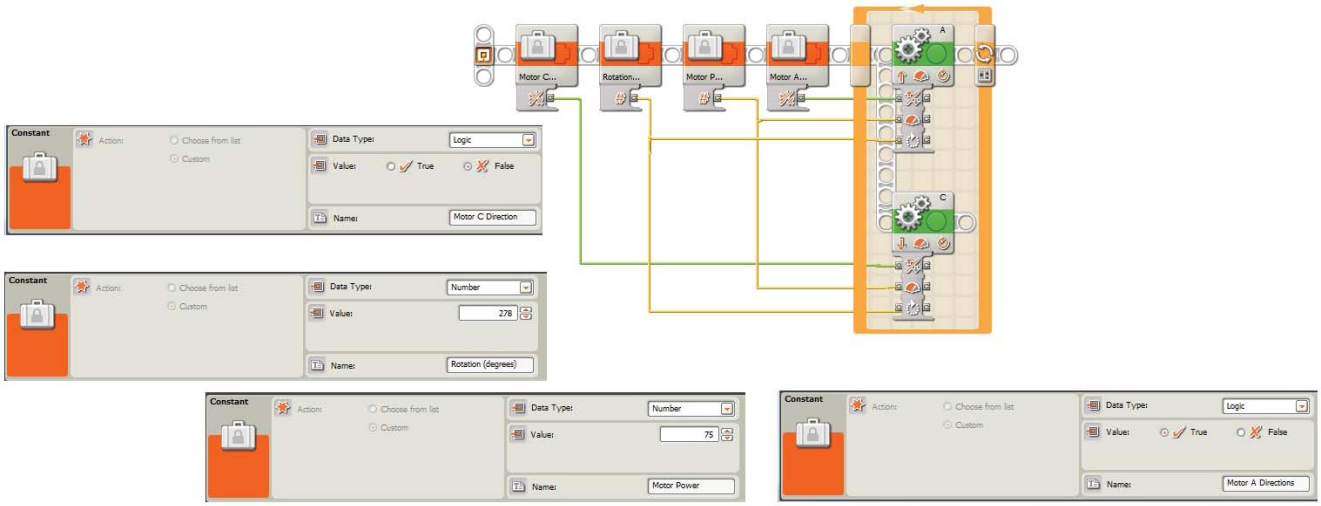
And again, we need to decide what we want to control, and what we want to work the same every time we run the program.

Control:

- **Direction Motor A**
- **Direction Motor B**
- **Power**
- **Duration**

Notice that to keep our turn balanced, we need to keep power and duration the same for both motors – so we don't need to control them separately.

Building the Turn MyBlock

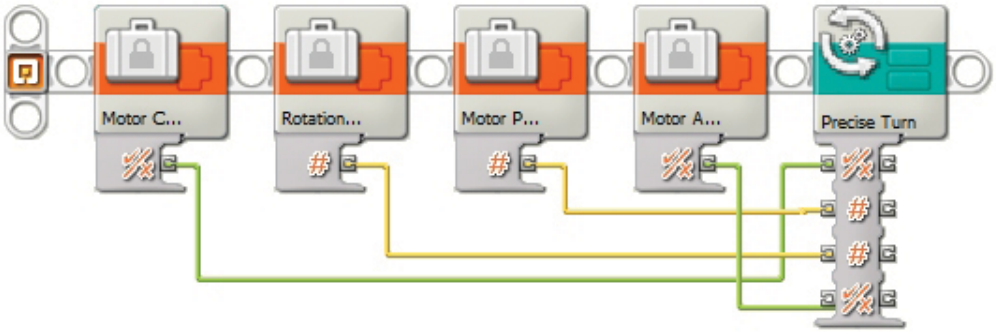


And again, we use the Constant blocks from the Data palette.

Power and Rotation constants control both A & C motors.

Motor A Direction and Motor C Direction are separate and opposite.

Building the Turn MyBlock

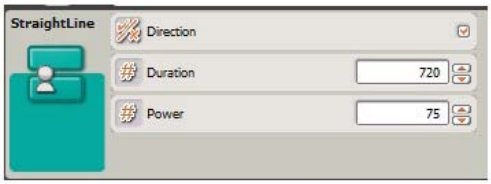


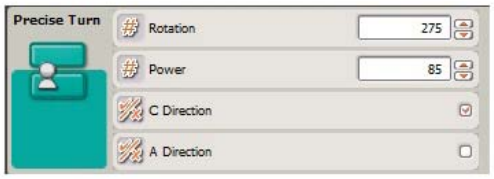
The NXT-G window after making the Precise Turn myblock.

Next we look at ways to use the myblocks as subroutines.

Putting it all Together

Loop count determines number of polygon sides





At its most simple, our two blocks can be filled in directly and the entire polygon program can be simplified to look like this.

On the Other Hand...

Power Level:
Sets Power for motors along sides and turns.

Could use different power for turns

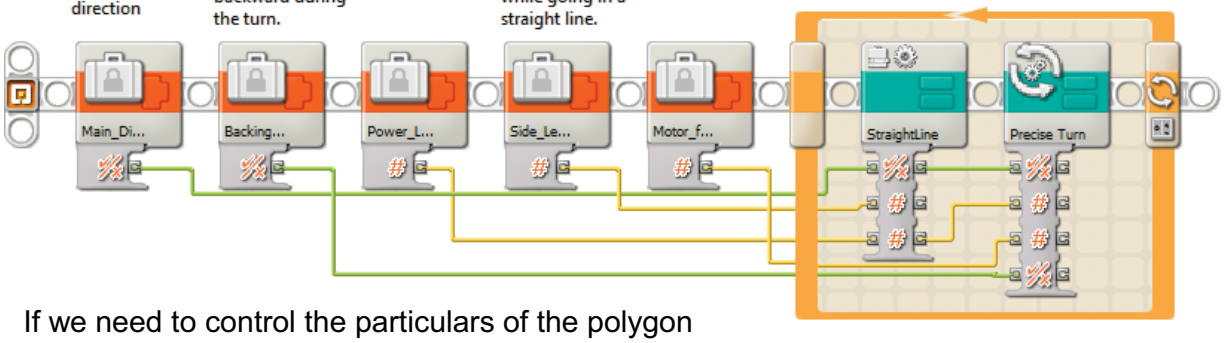
Motor for Turn (degree):
sets number of degrees for each motor to go forward and backward

Backing Motor:
sets a logical variable for the motor that runs backward during the turn.

Main Direction:
sets robot direction

Side Length:
Number of degrees to turn the motors while going in a straight line.

Loop Count determines the number of polygon sides



If we need to control the particulars of the polygon we can now send the controlling information into it and it will just work.

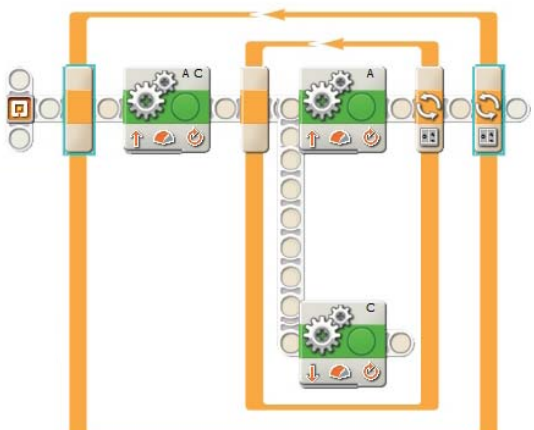
We can't collapse the whole effort down to one block, though, because the loop can't accept a variable for its counter.

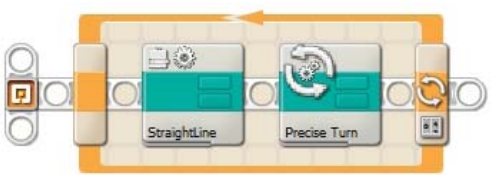
To Change direction of the robot's path along the polygon, swap the logic lines going into the Precise Turn Block

Conclusion

Regular polygon robot paths can be programmed with as few as five NXT-G blocks.

Myblocks can help simplify steps to keep the programs more clear and avoid repetition.

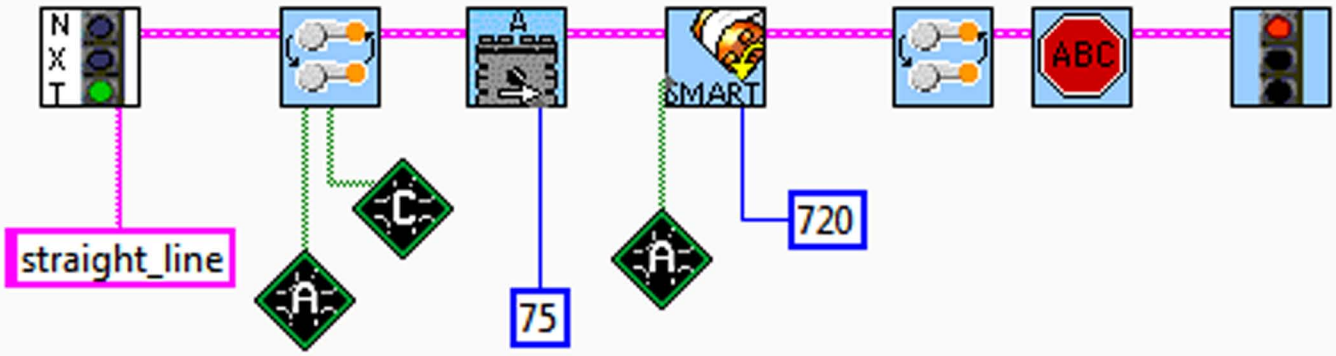




RoboLab Screenshots

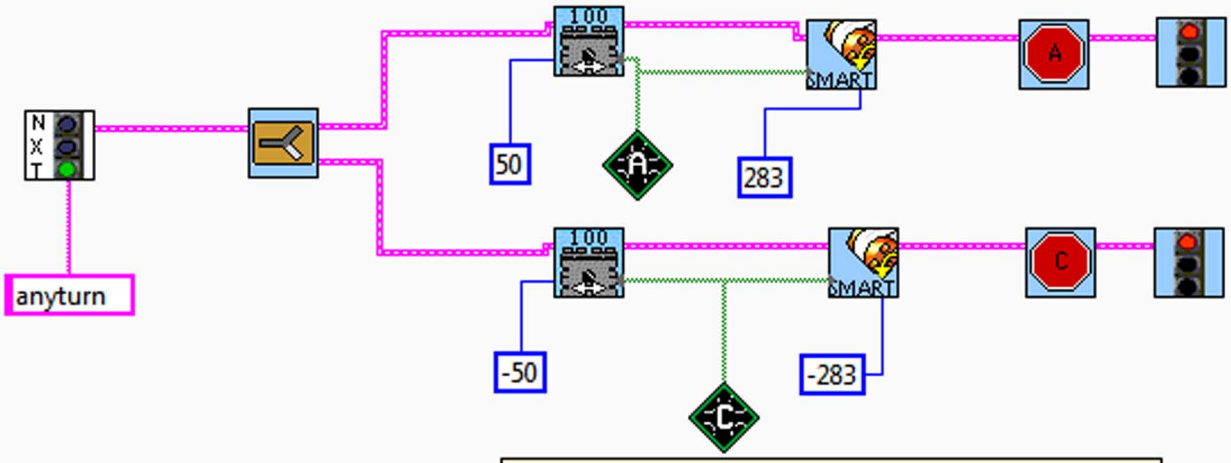
Straight Forward

This drives the robot forward
-- at the power assigned to the A Motor block
-- for the number of degrees assigned to the Smart Encoder Block

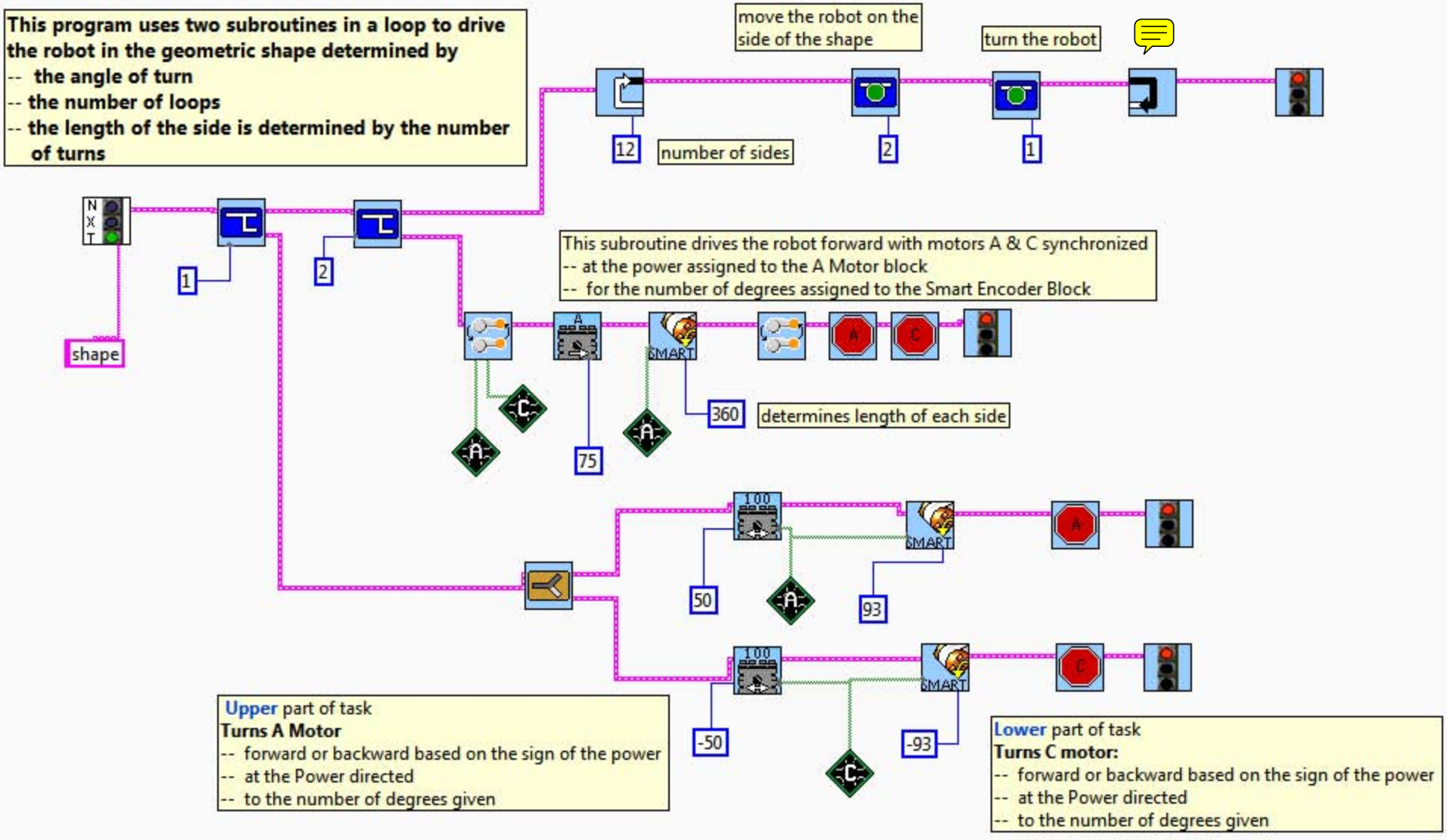


Precise Turns

Turns A Motor
-- forward or backward based on the sign of the power
-- at the Power directed
-- to the number of degrees given



Turns C motor:
-- forward or backward based on the sign of the power
-- at the Power directed
-- to the number of degrees given


This program uses two subroutines in a loop to drive the robot in the geometric shape determined by
-- the angle of turn
-- the number of loops
-- the length of the side is determined by the number of turns
move the robot on the side of the shape
turn the robot
12
number of sides
2
1
NXT
1
2
shape
This subroutine drives the robot forward with motors A & C synchronized
-- at the power assigned to the A Motor block
-- for the number of degrees assigned to the Smart Encoder Block
A
SMART
360
determines length of each side
75
100
50
93
-50
-93
Upper part of task
Turns A Motor
-- forward or backward based on the sign of the power
-- at the Power directed
-- to the number of degrees given
Lower part of task
Turns C motor:
-- forward or backward based on the sign of the power
-- at the Power directed
-- to the number of degrees given

Example Spreadsheet

Ungeared Robot Turn Calculator

b	**7**	Robot Wheelbase
Cturn	21.99	Turn Circumference
Cwheel	**7.1**	Wheel Circumference

Angle to turn	# of Degrees
0	0
5	15.49
10	30.97
15	46.46
20	61.95
25	77.43
30	92.92
35	108.41
40	123.89
45	139.38
50	154.87
55	170.35
60	185.84
65	201.33
70	216.81
75	232.3
80	247.79
85	263.27
90	278.76
95	294.25
100	309.74
105	325.22
110	340.71
115	356.2
120	371.68
125	387.17
130	402.66
135	418.14
140	433.63
145	449.12
150	464.6
155	480.09
160	495.58
165	511.06
170	526.55
175	542.04
180	557.52
185	573.01
190	588.5
195	603.98
200	619.47
205	634.96
210	650.44
215	665.93
220	681.42
225	696.9
230	712.39
235	727.88
240	743.36
245	758.85
250	774.34
255	789.82

b	**7**	Robot Wheelbase
Cturn	21.99	Turn Circumference
Cwheel	**7.1**	Wheel Circumference

Angle to turn	**# of Degrees**
0	0
260	805.31
265	820.8
270	836.29
275	851.77
280	867.26
285	882.75
290	898.23
295	913.72
300	929.21
305	944.69
310	960.18
315	975.67
320	991.15
325	1006.64
330	1022.13
335	1037.61
340	1053.1
345	1068.59
350	1084.07
355	1099.56
360	1115.05