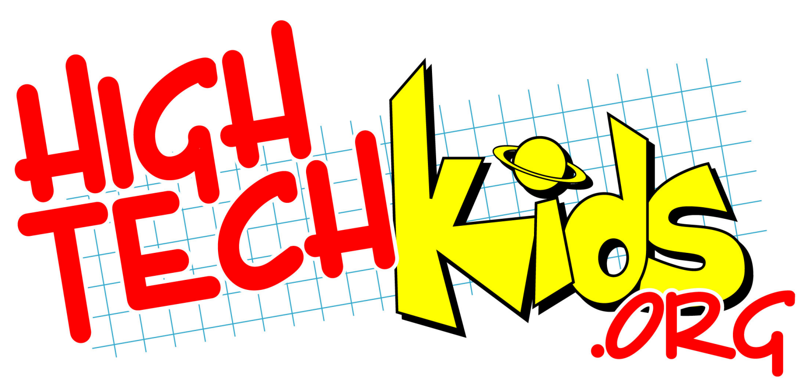
RIS version 2.0 Software Training Lab

Version 1.1
August 9, 2004

By
Joel Stone



www.hightechkids.org

About the Author

This document was written by Joel Stone, an FLL coach who wanted to pass on his team's hard earned wisdom about FLL programming. The document was edited by Doug Frevert and Fred Rose.

Copyright and Trademark Notice

© 2002-3 INSciTE in agreement with, and permission from FIRST and the LEGO Group. This document is developed by INSciTE and is not an official FLL document from FIRST and the LEGO Group. This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit
http://creativecommons.org/licenses/by-nc-sa/2.0/
or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

LEGO®, ROBOLAB, and MINDSTORMS™ are trademarks of the LEGO Group used here with
special permission. FIRST™ LEGO® League is a trademark owned by FIRST (For Inspiration and Recognition of Science and Technology) and the LEGO Group used here with special permission. INSciTE™ is a trademark of Innovations in Science and Technology Education.

INSciTE
PO Box 41221
Plymouth, MN 55441

A word about the Creative Commons License

High Tech Kids is committed to making the best possible training material. Since HTK has such a dynamic and talented global community, the best training material and processes, will naturally come from a team effort.

Professionally, the open source software movement has shown that far flung software developers can cooperate to create robust and widely used software. The open source process is a model High Tech Kids wants to emulate for much of the material we develop. The open source software license is a key enabler in this process. That is why we have chosen to make this work available via a Creative Commons license. Your usage rights are summarized below, but please check the complete license at: http://creativecommons.org/licenses/by-nc-sa/2.0/.

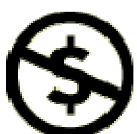
Attribution-NonCommercial-ShareAlike 2.0

You are free:

- to copy, distribute, display, and perform the work
- to make derivative works

Under the following conditions:

Attribution. You must give the original author credit.



Noncommercial. You may not use this work for commercial purposes.

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

Table of Contents

RIS version 2.0 Software Training Lab 1
Introduction to RIS programming 1
  Requirements to use this lesson document: 1
  The Hardware 1
  The Software 1
  Preparation 2
  Loading Firmware into RCX 2
  Program editing screen 4
Building your practice robot 5
Programming task 1: Move forward for 5 seconds & return. 7
  Requirements 7
  Command selection 7
  "End Program" command 8
  Download your program 10
  Testing your program 10
  Hardware changes 10
  Software changes 10
  Requirements changes 11
  Algorithm creation 11
  Topics learned in task 1: 11
Programming task 2: Move forward for 5 seconds & turn right 13
  Requirements 13
  Command selection 13
  Programming the turn 15
  Copy a string of commands 15
  Topics learned in task 2: 16
Programming task 3: Subroutines 17
  Requirements 17
  Command selection 17
  Scroll and zoom to see more of your program 17
  Problems with our current program 18
  Myblocks – how to dramatically improve the program 18
  Creating the first Myblock command 19
  Naming your Myblock commands. 19
  Saving your program 22
  Topics learned in task 3: 22
Programming task 4: Rotation sensor 23
  Requirements 23
  Structure of programming 23
  Opening (retrieving) a program 23
  Rotation sensor introduction 24
  Rotation sensor setup 24
  Manually measuring distance with the RCX 25
  Activating a robot's rotation sensor 25
  Measuring by moving the robot 26
  Programming with the rotation sensor 27
  Resetting the rotation sensor 28

Build a Fwd50 Myblock subroutine 28
Turning using the rotation sensor 28
Using variables to expand the robot's capabilities 28
Caveat of using rotation sensor 29
Topics learned in task 4: 29
Programming task 5: Forward to a black line, stop, turn right 31
Requirements 31
Light sensor 31
Light sensor programming 31
Build a FWD2LINE "Myblock" command 31
Changing light conditions 34
Additional notes 35
Which commands are used most often? 35
Which commands should be avoided? 35
Commands that are not provided by RIS 2.0 36
Wait for light sensor event OR touch sensor event 37
Debugging a program 38
Backing up your kid's program code 38
Error messages: 39

Introduction to RIS programming

Requirements to use this lesson document:

1. LEGO Robotics Invention System (The large box with the 700+ Lego parts – any version will work fine).

2. RIS (Robotics Invention System) version 2.0 software

3. Windows PC

4. White-topped table

The Hardware

The brain of the Robotics Invention System is the RCX, which stands for Robotics Command System. The RCX is a microcomputer, built into a yellow LEGO brick. The RCX uses sensors (such as touch and light) to take input from its environment. It can then process the data to make motors turn on and off.

To perform the lessons in this class, you will build a simple robot using a minimum number of parts.

The Software

FLL allows participating teams to use one of two programming languages. This class teaches the language RIS (Robotics Invention System). RIS programming language is formally called RCX Code but will be referred to as RIS in this document The other language that FLL allows is ROBOLAB.

Many people think that RIS is easier to learn than ROBOLAB. Neither language seems to have an edge in the competition. RIS will only run on a PC whereas ROBOLAB runs on PCs and MACs.

Your kids should be using RIS version 2.0. The earlier versions will severely handicap your kid's team. For only $20, RIS version 2.0 is available from SHOP.LEGO.COM, and is well worth the effort to acquire it.

When RIS or ROBOLAB programs are downloaded to the RCX, the programs are a type called "embedded code". Embedded water in a rock means the water is "stuck" inside the rock. Embedded code means that the program is "stuck" inside the RCX – there is no way to change the code after it is downloaded. Programs can only be modified on the PC!

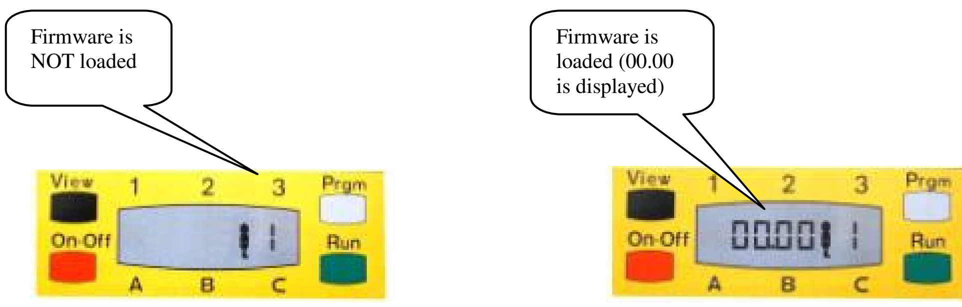
When kids program on the PC, they are entering commands in RIS or ROBOLAB. RIS and ROBOLAB use human readable commands. When downloading the program to the RCX, the PC converts the RIS or ROBOLAB commands to machine code, which the yellow brick can understand. This machine code does not make any sense to humans.

Preparation

Firmware is a special program that must be installed onto your RCX so that you can program the RCX brick.

This preparation step is only required to be performed once when the RCX is new. If the RCX loses its firmware in the future for some reason (such as changing batteries too slowly), or starts behaving badly, you may want to reload the firmware again.

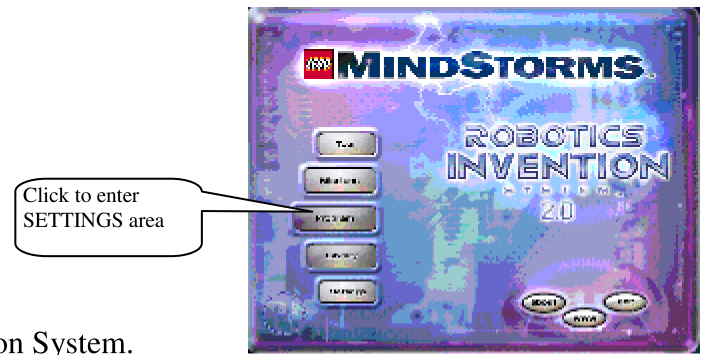
The RCX brick must be set up properly prior to downloading your team's programs into it. The RCX brick must contain the proper version of firmware in order to proceed. The following photos show the RCX's screen without and with firmware loaded.



Unfortunately, even if you see that firmware is loaded, it still may be the wrong version of firmware.

Loading Firmware into RCX

To load firmware into the RCX or verify the firmware version, start the RIS program and go to the settings screen.



1. Start up the Robotics Invention System.
2. Select **New User.**
3. Type your name and then **Enter**.

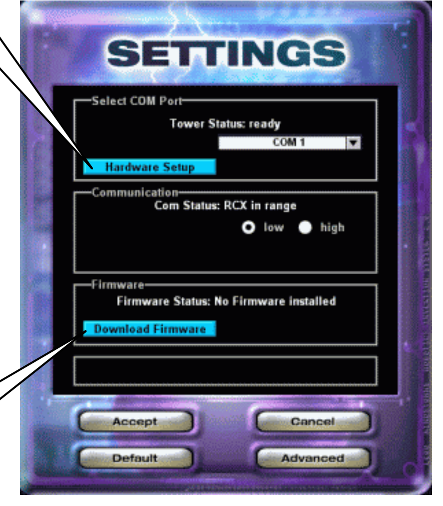
Watch the video for awhile, then press "**enter**" to move on.

You should now see a menu starting with the **TOUR** option

The Settings screen

Click to set up IR tower

The “Settings” menu option can help with all sorts of issues that may arise.



Is firmware version 2.0 installed? If not, click Download Firmware and follow instructions.

Connect the IR tower to your computer and turn on the RCX yellow brick. Make sure the switch on the front of the tower is moved to the left. That is the low power setting.

Point the RCX towards the tower – the black lens on the RCX should be facing the tower.

Click “Settings” on the PC. You will see this screen:

At this point, the PC and the RCX will communicate with each other.
This will only occur if the green light appears on the front of the tower.

If the green tower light turns on then skip to the next sentence. If the tower’s green light does NOT light, click “**Hardware setup**” on the SETTINGS screen to set up your IR tower.

If the RCX has firmware loaded, the PC will verify the firmware version. If the RCX firmware is missing, the PC will prompt you to download the firmware to the RCX. If required, go ahead and download the firmware.

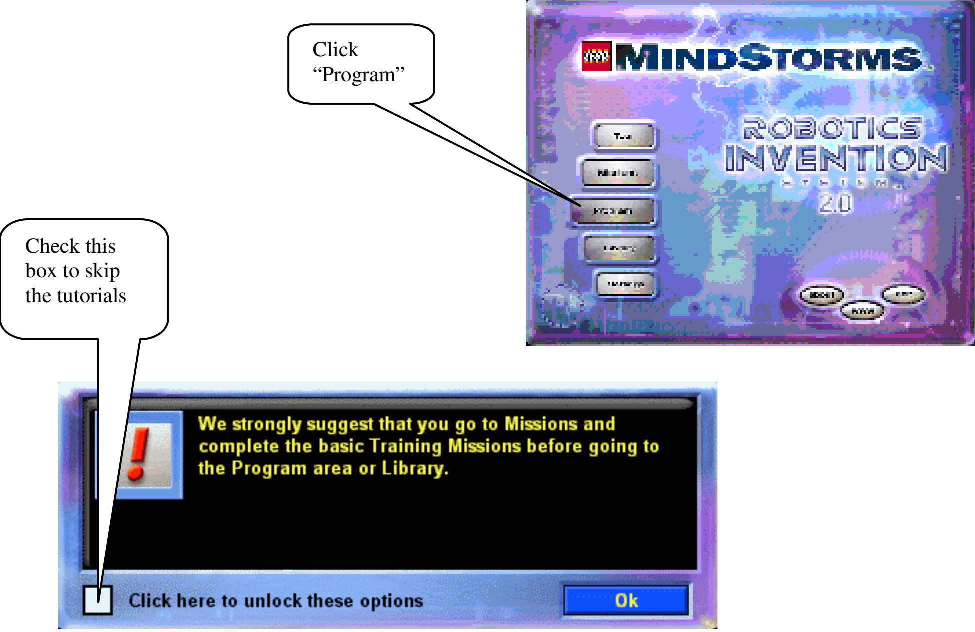
You should **verify that firmware version 2.0 is loaded into the RCX**.

You will also notice the battery level of the RCX displayed on the SETTINGS screen, which will be helpful in the future.

If the Firmware doesn’t automatically install for some reason, or you need to reinstall firmware in the future:

1. Place the RCX 6 inches from the tower.
2. Turn on the RCX and point the RCX black window towards the tower.
3. From the **SETTINGS screen,** click “**Download Firmware**”
4. Once firmware is installed, you are ready to start programming!

Program editing screen

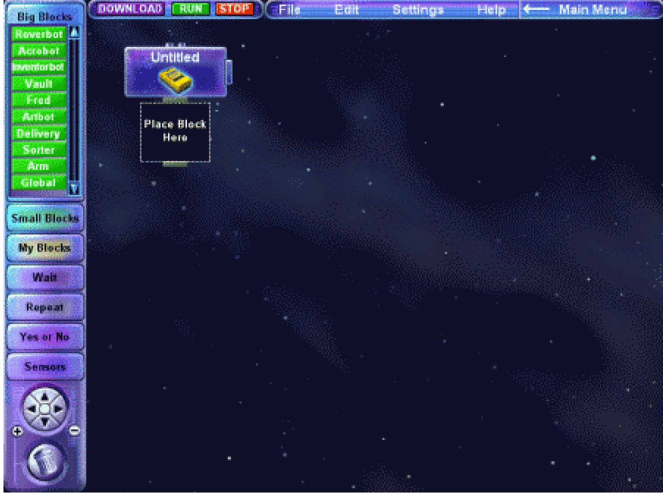


then click OK. This enables you to program without going through the tutorials. (However, the tutorials are worth going through!)

If the window doesn't appear, that is OK.

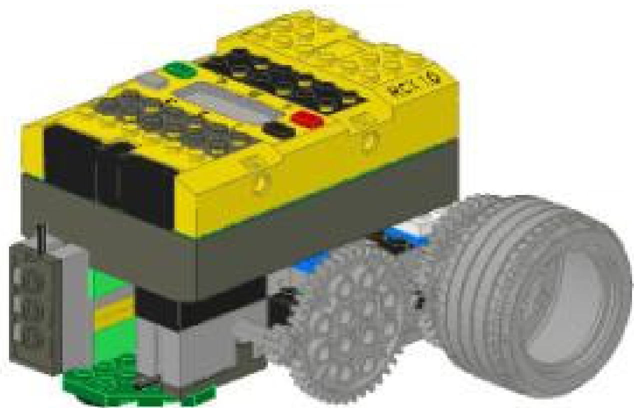
Click "**Program"** again and then click "**Freestyle".**

You should now see the main programming screen:



You are now ready to begin creating a program.

Building your practice robot



Before you can start programming, you must build a robot. We have created a simple-to-build robot to use for this class. The following requirements were met when designing this robot:

1. Build time less than 10 minutes
2. Simple design (too simple to be of much use in the competition)
3. Minimum possible LEGO parts used, but still support use of rotation and light sensors for learning.

The parts list and build diagram are located in a companion document.

Programming task 1: Move forward for 5 seconds & return.

Time: 30 minutes (20 minutes to create and run program + 10 minutes Q&A)

Requirements

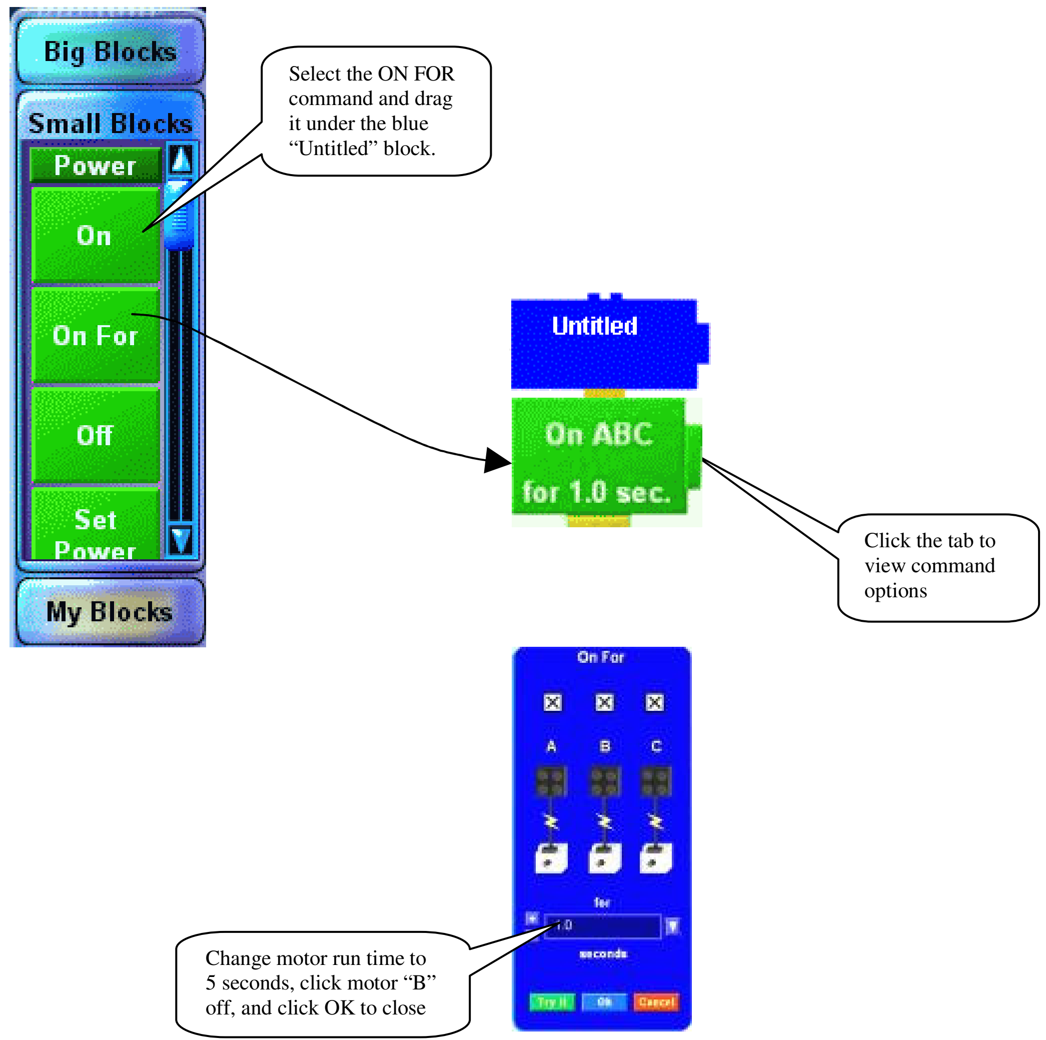
The first step in programming involves stating what needs to be accomplished. For this mission, we want to move forward for 5 seconds. Given these requirements, the kids need to think of how to accomplish the task. With the robot that we are using, simply turning on both motors for 5 seconds will meet our objective.

Command selection

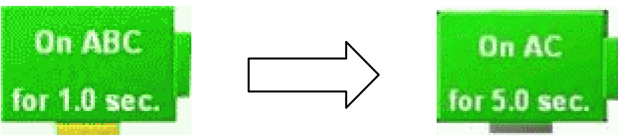
To program this first task, we simply want to move forward for 5 seconds, stop, and return.

To move forward, we need to turn on the motor on port A and port C for 5 seconds.

The commands we will use are located in the SMALL BLOCKS menu, in the POWER section.



Notice that the ONFOR command changed from:



Motor "B" is no longer being turned on, and the time is now 5 seconds instead of 1 second.

"End Program" command

It is a good idea to always end your program with the "End Program" command. Go ahead and grab that command from the Advanced group in the Small Blocks menu. The "End Program" command stops the program from running on the RCX. Without this command, the kids must press the RUN button to stop the program, which gets very confusing during a high pressure tournament.

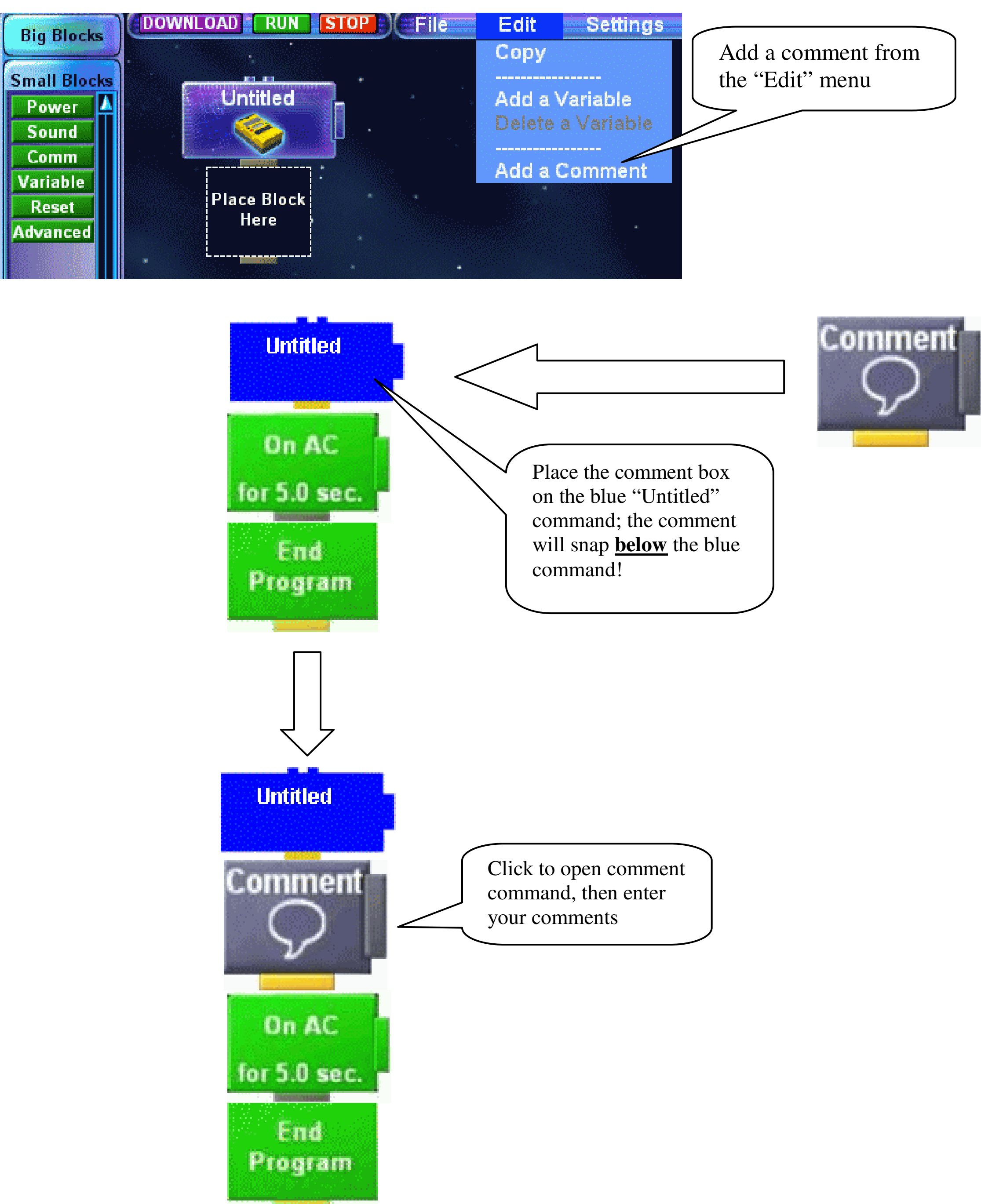
Pressing RUN should only be needed to start the program. Without the END command, the kids need to press the RUN button twice, once to start the program and once to end the program.

Your program should now look like the following:

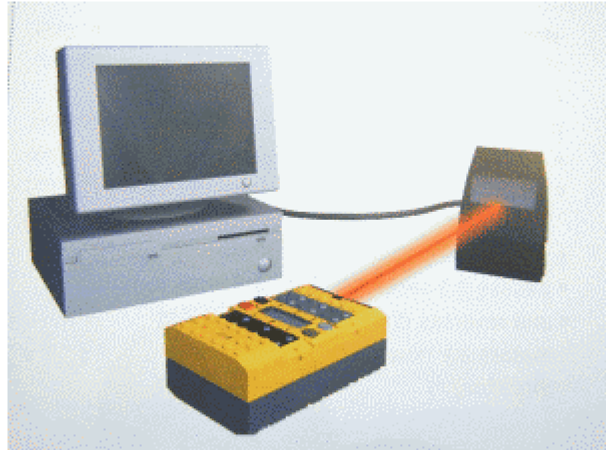
Comments

One of the most important tasks in the creation of a program is to document what the program does. This can be accomplished in RIS by adding a comment.

Add a comment as follows:



To download the finished program to the RCX, assemble the parts as in the drawing below:



Download your program

Click "Download" at the top of the screen to transmit the program to the robot.

Program development now moves from coding to testing.

Testing your program

Press the green RUN button on the robot to test your program.

Does it run forward for 5 seconds, and then stop?

If it works properly, congratulations.

If one motor or both motors move in the wrong direction, how can the problem be corrected?

With robots, there are often two methods of making changes: one method is making a hardware change; the other is a software modification.

Hardware changes

If a motor is running in the wrong direction, how can it be reversed by modifying hardware? The motors on the robot run on DC current, which means that they can be reversed by changing the polarity. Try rotating the connector to change the direction of the motor.

Software changes

Can you find a software command to set or change the direction of the motors?

Requirements changes

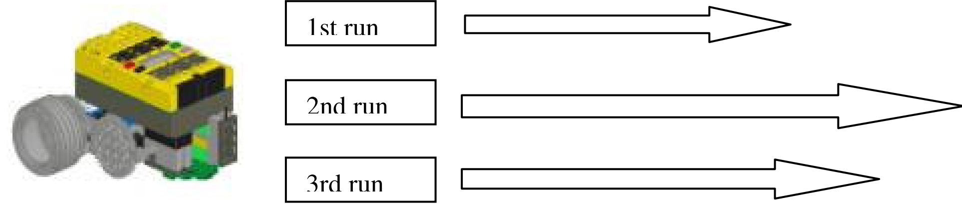
After running the program and watching it perform, your kids will undoubtedly want to change either the requirements or the method they used to meet the requirements.

Try changing your program to return the robot to its starting point after moving forward.

Algorithm creation

When you made decisions about how to approach the requirements, you created an algorithm, or series of steps to meet the goals of the task. These steps were turned into program commands.

Try running your robot three times starting from the same spot.



When your robot moved forward for 5 seconds, did it cover the same distance each time you ran the program?

As the batteries wear down, will the robot travel the same distance each time the program is run?

Think about the FLL challenge: is it often important to move the same distance each time the program is run?

When is it important to move the exact same distance each time the program is run?

- During a turn would be a good example – you don't want an overturn or underturn.
- Another example would be when approaching an object to be retrieved. You don't want to knock over the object, and you also don't want to come up short!

Try to think about other methods that you could use to travel a known distance more accurately and repeatably.

Topics learned in task 1:

Congratulations! You have successfully programmed your robot, and have learned the following topics in programming task 1:

- programming editor navigation
- selection of commands
- variety of commands available
- how to add comments
- thought processes required to move your robot forward
- housekeeping tasks required in programming
- sequencing of commands
- shortcomings inherent with program as designed
- distance traveled using time is not accurate

Programming task 2: Move forward for 5 seconds & turn right.

Time: 30 minutes (20 minutes to create and run program + 10 minutes Q&A)

Requirements

This lesson builds upon the previous program. After moving forward for 5 seconds, we now want the robot to perform a 90 degree right turn. To perform these requirements, we will simply turn on both of the motors, but one motor will be run in reverse. This will cause a rotation of the robot, resulting in a turn.

Command selection

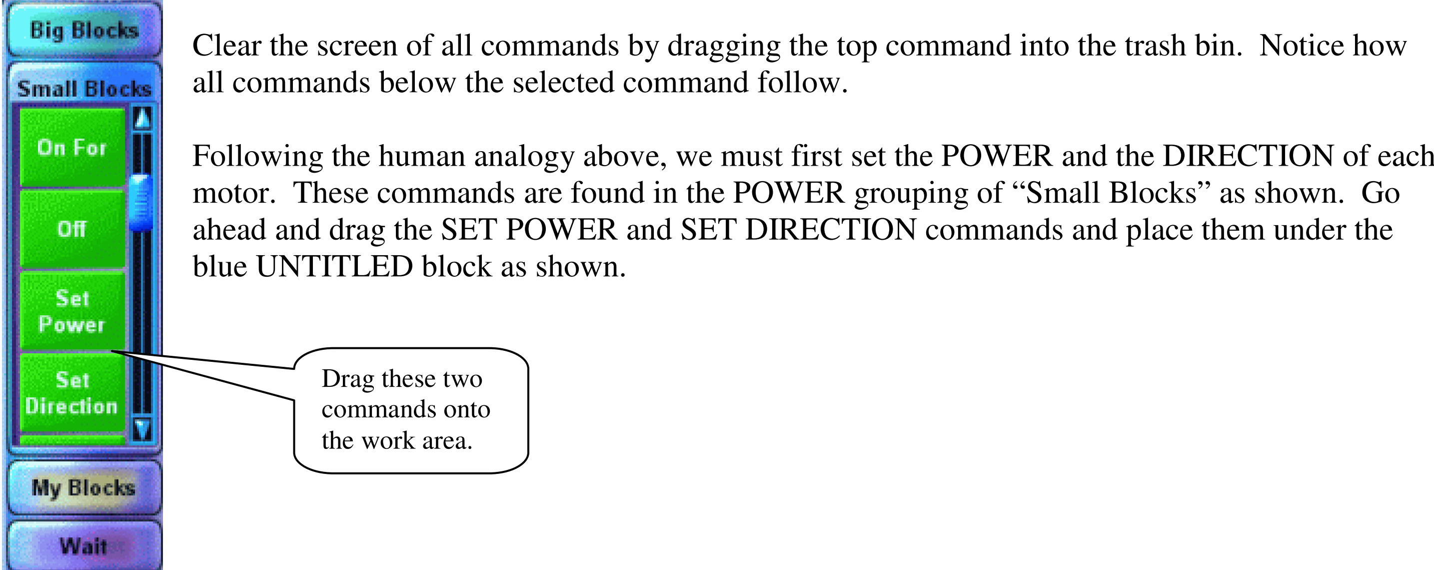
To determine what a robot must do to accomplish a task, it is often useful to develop analogies to share with your team. This method will relate something that they don't understand with something that they are already familiar with.

For example, lets create an analogy about starting our robot moving forward. The analogy will be with a person walking. Before starting to walk, what must a person think about? Several "set-up" items should come to mind.

1. What direction am I going to walk?
2. How much power am I going to exert while walking?
3. When will I begin my walk?
4. What will cause me to stop walking?

These steps translate directly into robot commands. The first lesson was over-simplified, but worked OK. In this lesson we will be more thorough in describing the robot's travels.

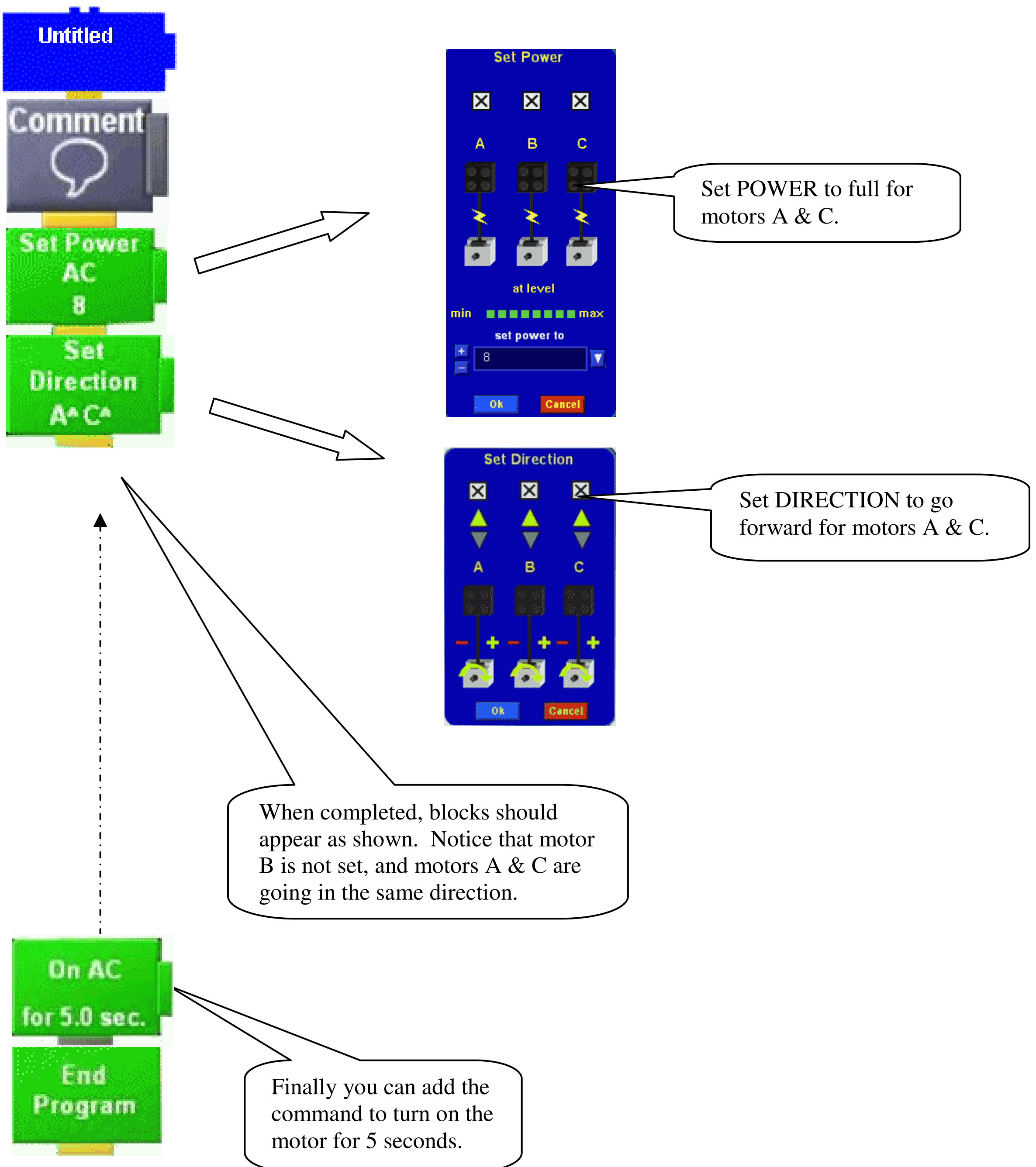
Clear the programming screen



Clear the screen of all commands by dragging the top command into the trash bin. Notice how all commands below the selected command follow.

Following the human analogy above, we must first set the POWER and the DIRECTION of each motor. These commands are found in the POWER grouping of "Small Blocks" as shown. Go ahead and drag the SET POWER and SET DIRECTION commands and place them under the blue UNTITLED block as shown.

After placing the comment at the start of the program, and adding the two commands, go ahead and click the tabs on the side of each command block.



Programming the turn

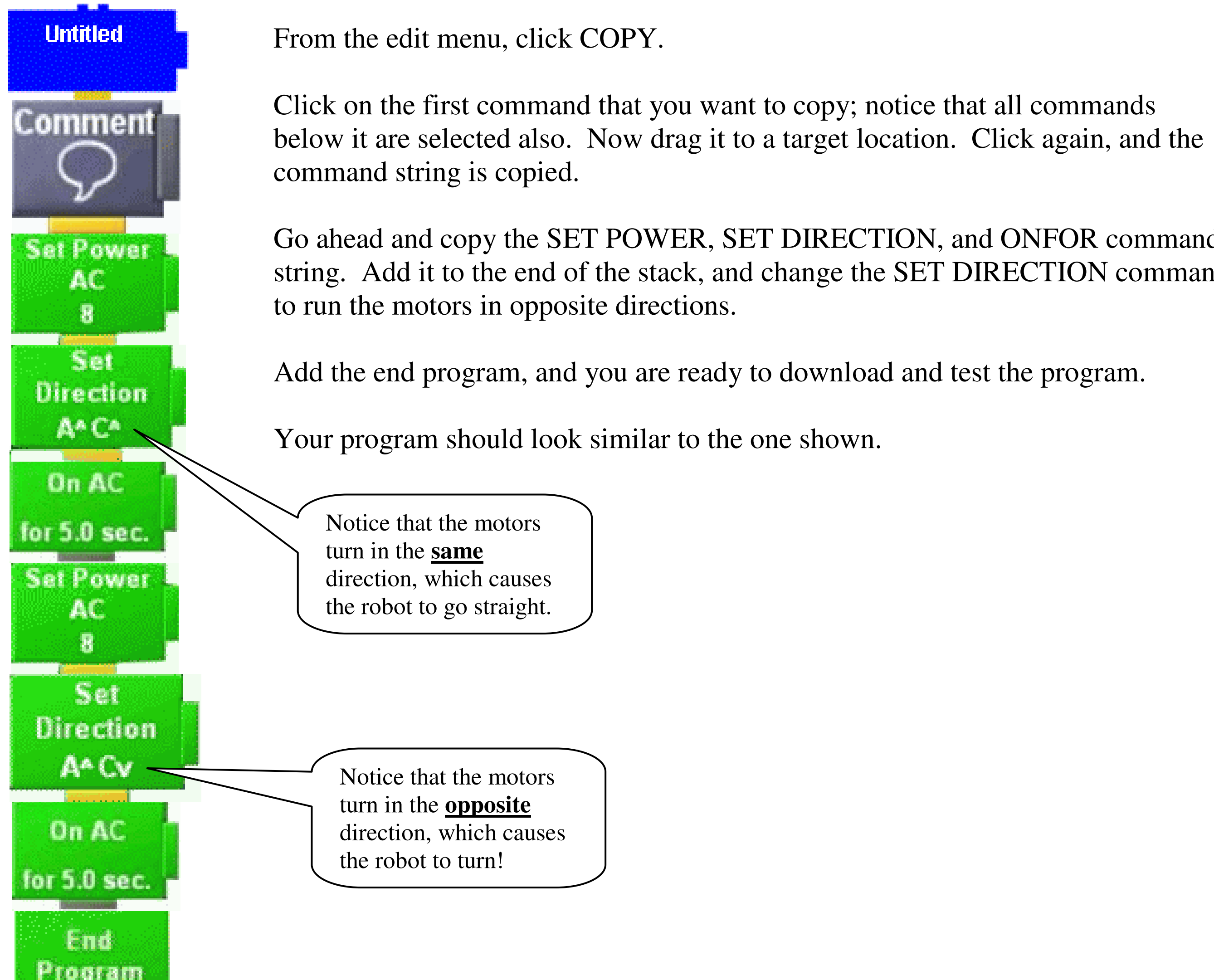
To program the robot to turn, we want the motors to spin in the opposite direction for a short time. Can you think of a way to do this?

We will use the exact same commands as above, but will simply change the command options. Which options need to be modified to perform a turn?

The only options that we need to change are the direction of one motor, and the time that the motors are turned on for.

Copy a string of commands

First, we need to copy the commands that we already have, and bring them to the bottom of the stack. To copy the string of commands, we could grab new commands and add them to the stack. An easier way is by using the copy command.



From the edit menu, click COPY.

Click on the first command that you want to copy; notice that all commands below it are selected also. Now drag it to a target location. Click again, and the command string is copied.

Go ahead and copy the SET POWER, SET DIRECTION, and ONFOR command string. Add it to the end of the stack, and change the SET DIRECTION command to run the motors in opposite directions.

Add the end program, and you are ready to download and test the program.

Your program should look similar to the one shown.

Testing the new program

Now it is time to download the program and run it.

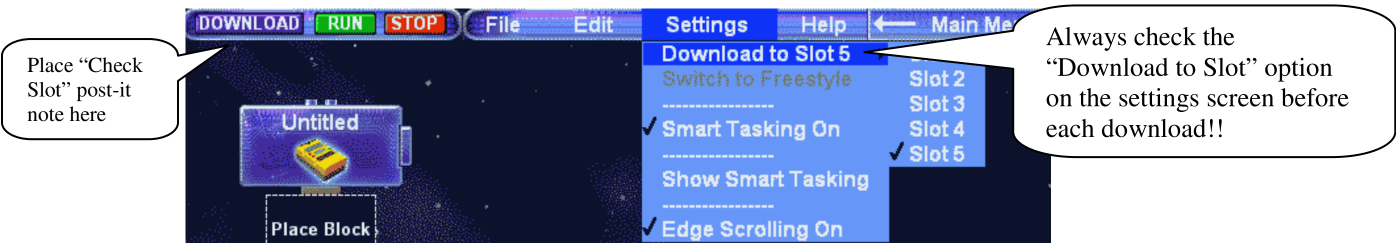
Does the program perform as expected? Does the robot make a 90 degree right turn? Probably not.

What can be done to make the turn closer to 90 degrees?

Did the robot turn right, or did it turn left by mistake? How can you correct this problem?

Program slots – pitfall to avoid!!

The RCX is capable of holding 5 programs at a time. Each program is downloaded to a program "Slot". Prior to downloading a program, always check which slot it will be loaded into. This is done with from the "Settings" menu option on the programming screen as shown.



I recommend cutting a tiny portion of a post-it note and placing it over the "Download" command on the screen. Write on the note "Check Slot".

The PC will download to the same slot as the prior download. This is OK if you are working on program 1 and downloaded to slot 1 recently. But if your kids are making an emergency change during a tournament to program 5, and your kids forget to change the slot from the prior download of program 1, guess what? Program 1 no longer does what it should – it now contains program 5. The kids will be very disappointed when they run the mission and it doesn't work.

Best bet is to simply check the settings prior to each download. This will get the kids in the habit of verifying which slot has which program!

Ideally, RIS should remember which slot each program belongs in, but unfortunately it doesn't.

Topics learned in task 2:

Congratulations! You have learned more about the following topics:

- difficulty with turn based on motor run time
- demonstrate with people – must set power and direction before walking
- use metaphors to help kids see concepts
- copy command strings
- downloading to various program slots

Programming task 3: Subroutines

Time: 30 minutes (20 minutes to create and run program + 10 minutes Q&A)

Requirements

This lesson builds upon the previous program. After moving forward for 5 seconds and turning right, lets run the robot in a pattern to complete a square. To perform these requirements, we will simply duplicate the code from the last lesson three times!

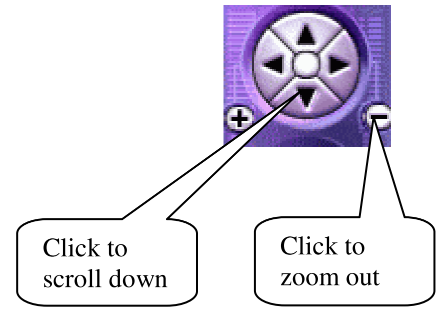
Command selection

You should be able to complete the square pattern on your own. What is the easiest way to do this?

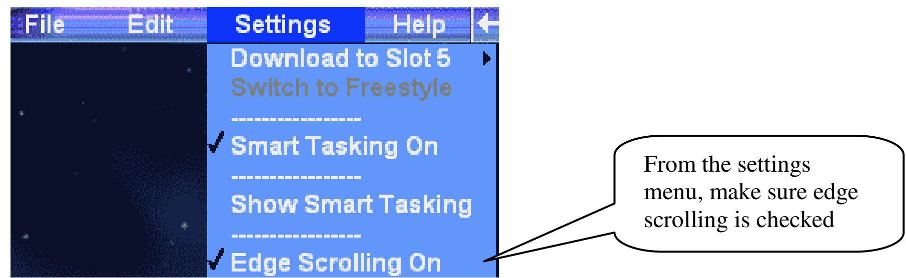
Use the COPY function to grab all of the commands except the END PROGRAM, and paste them to the end.

Scroll and zoom to see more of your program

You can scroll up and down to see more of your program using the scroll button. The zoom + and – are located near too.



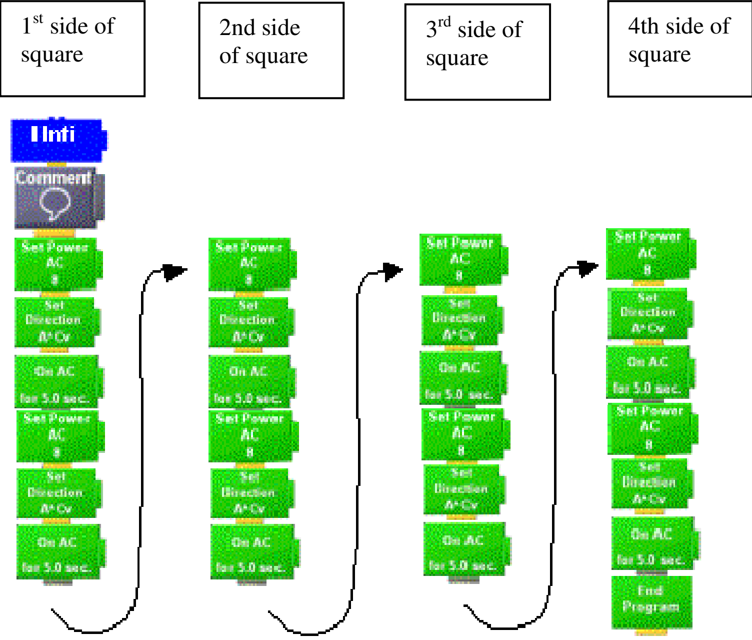
The better way to scroll is to turn on scrolling. This only needs to be done once – RIS seems to remember this setting. Then you can simply move the mouse pointer down and the screen will scroll for you as you would expect.



If scrolling doesn't work by dragging the mouse down, try clicking anywhere on the black background!

Lets get back to our program:

Now what do we have? A big mess! This is where many teams get themselves into trouble – the command strings get so long and complex so quickly, that no one can see the overall picture! Here is what your program should look like (it is displayed more horizontally here):



Problems with our current program

- Imagine hundreds of commands filling up the programming screen as the robot travels around the mission board. How does one keep a sense of sanity with such complexity?
- What if you want to modify the right turn portion of each leg, for example add a beep after each right turn. How many duplicate changes must you make?
- If you want to move a portion of code around, it is easy to grab the wrong chunk of code.

Myblocks – how to dramatically improve the program

There is one solution that quickly addresses all of the above problems – using subroutines! In RIS, subroutines are called “Myblock” commands. These are new commands that you create!

Let’s create a Myblock command to go forward for 5 seconds, and another one to turn right. The idea is to build a Myblock command for each robot function.

Creating the first Myblock command

Click on the My Blocks button, and drag the "Create new My Block" command onto the screen. Place it anywhere on the dark screen by clicking again. You are now forced to name the Myblock.

Naming your Myblock commands.

Naming is very important. If done properly, it will help your kids tell at a glance where they are and what the program is up to. Choosing poor names makes programming much more difficult. Choosing names that are too long shows up as a small font, and makes it difficult to see the names.

Think of some of the basic motion functions that the FLL robot may need to do for the challenge:

1. move the robot forward until it hits a wall, then stop;
2. move forward and stop at a black line
3. move the robot forward a certain distance
4. turn right 90 degrees

Use 10 or less characters to keep the font size large. After 10 characters it gets too small to read.

I suggest using "action + to + target", but this can be varied. To name the examples above, I might use the following names:

1. Fwd2Wall
2. Fwd2Line
3. FwdDist
4. TurnRight

As you can see, you have to be creative. But some naming consistency makes sense.

Some poor examples might include:

- "Fwd5sec" (when changing to 6 seconds, you don't want to have to change the name);
- "GoToTallTower" (too long, and probably can't re-use such a specific subroutine).
- GoStraight (no sense of direction – can't tell if the robot will move forward or backward).

Back to our square pattern. Lets create two Myblock commands. One will forward for 5 seconds. The next one will turn right.

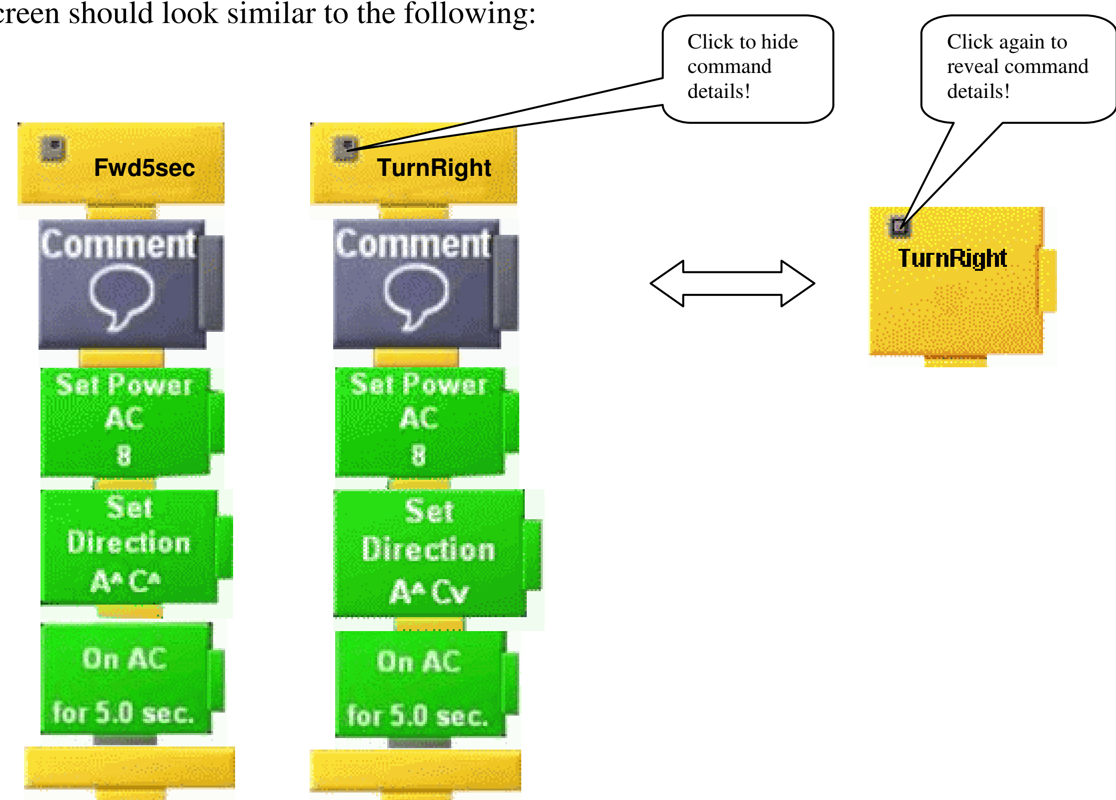
Name the first command "Fwd5sec", and name the next one "TurnRight".

Grab the green blocks that make up the straight travel portion of the program, and snap them into the "Fwd5sec" Myblock.

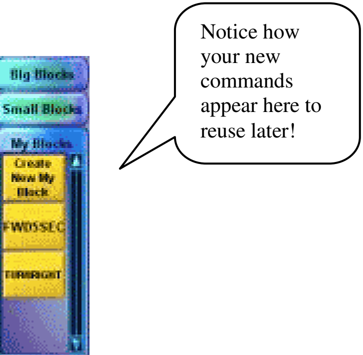
Grab the green blocks that make up the turn portion of the program, and snap them into the "TurnRight" Myblock.

Trash any other commands on the screen except the "End Program".

Your screen should look similar to the following:



Notice how your newly created commands appear under the Myblock commands:



Shrink the new commands by clicking on the gray square in the yellow box. This action hides the complexity of the new function so you don't have to see it unless you want to. Open the Myblock by clicking on the gray square again to see the green commands, then hide it again.

Put the square program back together again using the newly created Myblock commands.

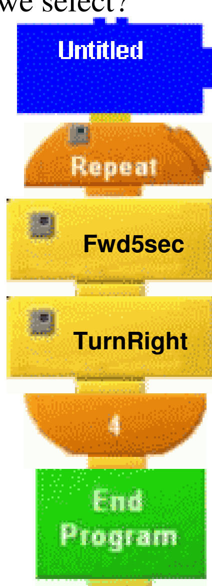
You should end up with the following program:

This program looks better, but notice that the commands repeat. Lets simplify the program using the "Repeat" command. Open the Repeat menu. Which Repeat command should we select?

Use the "Repeat For" command to drive the robot in a square.
Click open the "Repeat" command options,
and enter 4 for the number of repetitions.

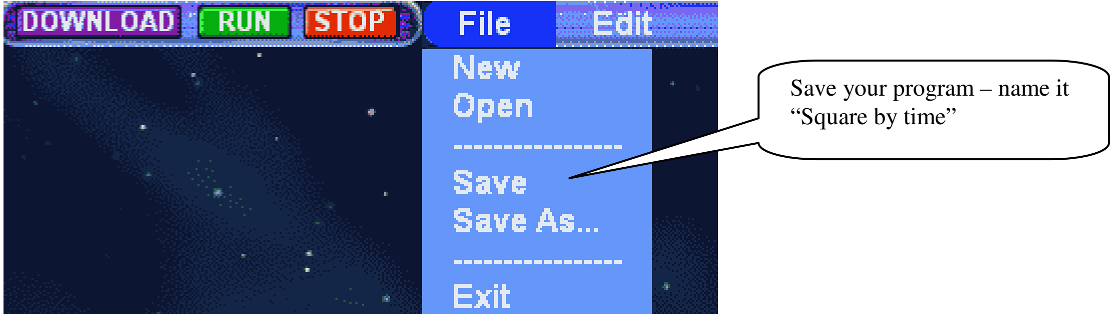
Your program should look as shown.

This is about as easy to read as the program can be. Can you see the defference between this program and the versions above that perform the same routine? See the simplicity of the code? It is very easy to read. The complexity of the code is hidden well, but can be viewed easily if desired by opening the Myblock commands.



Saving your program

Go ahead and save your program using the File – Save menu option.



Notice how the top blue block changes to the program name.

Topics learned in task 3:

Congratulations! You have learned more about the following topics:

- How to set programming window to scroll
-
- Why use subroutines:
    1. Most important reason: reduce a program's complexity
    2. Create subroutines to hide information so you wont have to think about the details
    3. Think about the subroutine when you are writing it, but use it later without knowing about its internal workings
    4. Avoiding duplicate code
    5. Promotes code re-use
    6. Isolates complex operations
    7. Etc etc
- create Myblock subroutine
- importance of names
- importance of hiding information
- difficulty with turn based on motor run time
- strings of individual commands is difficult to follow

Programming task 4: Rotation sensor

Time: 30 minutes (20 minutes to create and run program + 10 minutes Q&A)

Requirements

In this lesson, we will use the rotation sensor to measure distance traveled more accurately. This should result in a robot that gives a more repeatable performance run after run.

Structure of programming

You should now have a good idea of how to structure a program. We want to use well-named subroutines to create building blocks of code fragments that perform specific tasks.

As you have probably noticed, locomotion using motor run time will result in limited performance What we have to cover now is building some really useful routines. Turning motors on for a specific amount of time is of limited value in FLL – it will get the kids started but isn't going to help with most of the missions.

Opening (retrieving) a program

Big tip: NEVER open a program from the programming screen. There is a severe bug in the RIS software which causes Myblocks to be renamed when opening a program this way. ALWAYS EXIT from this screen back one screen, then come back into the programming screen to load a program

IF YOU EVER ENCOUNTER the RIS dialogue box "You must rename this Myblock", you must immediately kill the program using Ctrl-Alt-Delete". This is the only way out of the problem.

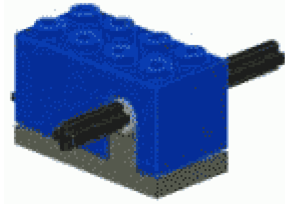
This only seems to occur when retrieving a program, so you shouldn't lose any work forcing this exit.

You do NOT want to be forced to rename Myblocks that are used in other programs – it is best to leave them be! If you are being forced to rename Myblocks, you are somehow loading your program in a way RIS doesn't like.

If you need to rename a Myblock because you WANT to rename it, that is a different story, and can be done. Be aware to go into any other programs that used it and make sure it picks up the new name properly!

NEVER use this OPEN command

Rotation sensor introduction



Rotation sensor

Why use a rotation sensor? As you have probably noticed, positioning a robot based on motor run time leaves a lot to be desired. Any of the following can cause the robot to alter its path based on motor run time:

- If the Lego parts shift a bit, and there is now a little more friction between gears;
- The battery has more or less power output than the prior run;
- The surface friction changed by warming up, cooling down, etc;

These uncontrollable issues make time a poor way for the kids to navigate their robot.

Think of giving a friend driving directions to Chicago. Would you say "drive you car for 25,395 seconds, turn right for 10 seconds, then drive for 250 seconds to a restaurant"?

They would not get near the restaurant that you had in mind! It probably would land them in a different city, and it is likely that after the first turn, they turned off the road and onto a farm!

A more accurate instruction set would be "drive 314 miles, then exit at the water tower. Turn right, and go 2.4 miles to the restaurant". Your friend would most likely end up at the same place that you had planned on.

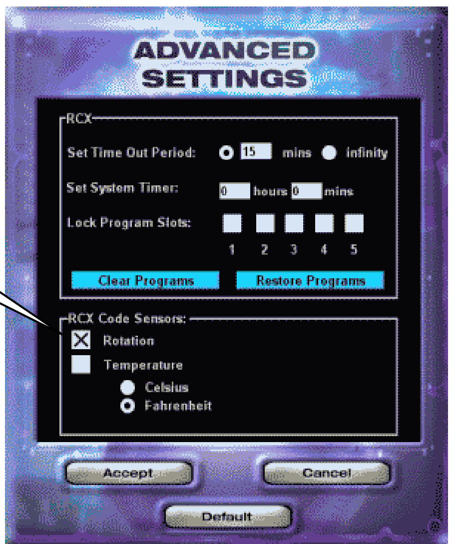
A rotation sensor can behave as an odometer for the Lego robot. It can measure the number of rotations that an axle turns. If the axle is somehow connected to a wheel, it will quite accurately measure distance traveled.

Rotation sensor setup

For some reason, RIS requires that you tell it that you want to use a rotation sensor. This is done from the SETTINGS screen (NOT the settings menu!)..

From the SETTINGS screen, click ADVANCED. You should see the following screen:

Click on the Rotation sensor box to place an "X" in the box. This will allow the rotation sensor to be used.



Note: this only has to be done one time and never again!

Click ACCEPT and go to a blank programming screen.

Manually measuring distance with the RCX

It makes a good demonstration to measure a distance with the robot using the rotation sensor. And, this method will be used often to measure distances. Why bother converting from inches to rotations and back many times? Simply use the robot to take the measurements, then your kids will know exactly how far it must travel!

Before we can use the robot to measure, the RCX computer needs to have the rotation sensor activated.

Is this the same thing we did in the SETTINGS screen above? No! The settings screen change allowed us to use rotation sensor commands to build a program. Now that we can add a rotation sensor command to our program, it must be downloaded to the RCX in order to activate the rotation sensor.

Activating a robot's rotation sensor

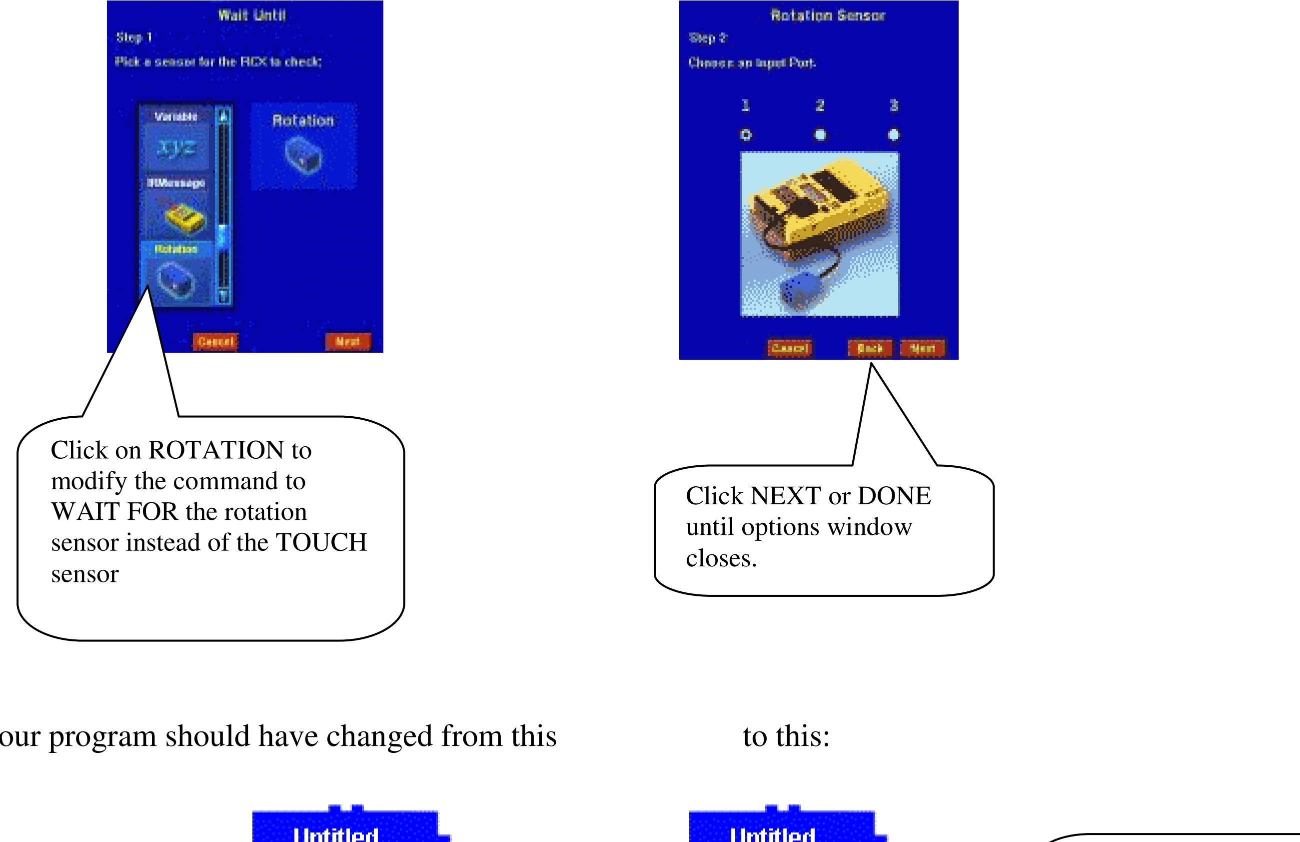
To activate the rotation sensor on the robot, we must build a program that uses the rotation sensor. This can be a very simple program, such as one command to wait for the rotation sensor to reach a certain value.

To add a rotation sensor, first drag a WAIT UNTIL command into a new program:

The WAIT UNTIL defaults to waiting for the touch sensor on RCX port 1.

We need to change the command to WAIT UNTIL the ROTATION sensor reaches a certain value.

Open up the options for the WAIT UNTIL command and scroll down to the ROTATION sensor. Click on the ROTATION sensor to see the following:



Your program should have changed from this to this:

Untitled
Wait Until
Touch 1

Untitled
Wait Until
Rotation 1

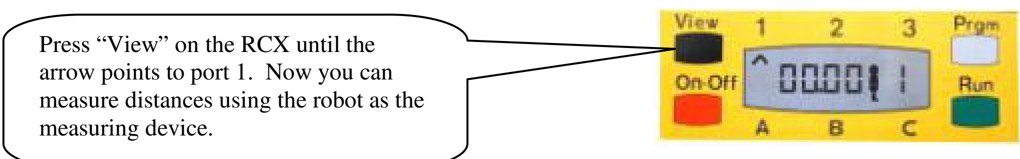
Your program should be waiting for the Rotation sensor instead of the Touch sensor!!.

Now download this simple program to the RCX and we can measure distances with it!

Measuring by moving the robot

Any sensor can have it's current value displayed on the RCX window. After downloading a program containing a rotation sensor command, you are ready to begin measuring.

Press the view button until the arrow is pointing to the sensor port that you want to monitor. For your robot, press "view" until the arrow is pointing to port 1.



Give it a try! Press the RUN button on the robot to execute the program and activate the rotation sensor. As you move the robot around, does the RCX window display the rotation counter value?

Programming with the rotation sensor

Now lets turn our attention back to programming. Lets say that we hand-measured a distance of 50 that we would like the robot to move forward.

How can we program this using RIS?

First, think about how to walk 50 feet. Here are the first few commands to walk 50 feet:

1. set the zero mark at where you currently are standing
2. point your legs in a certain direction
3. set the power that you legs will use
4. walk forward

Now the tricky part: what is the next command that you would give your legs?

If you think about it, the next command is “WAIT”. “Wait” seems like it would indicate stopping, but it does not. “Wait” means to continue doing whatever you are doing, until an event occurs.

If your goal is to walk 50 feet, the event would be your eyes sensing that you have reached 50 feet. At that moment, you would issue a command to your legs to stop walking.

Here is a series of commands that might complete the above sequence for the Lego robot:

1. reset the rotation sensor to zero
2. set motor direction
3. set motor power
4. turn on both motors
5. WAIT until the rotation sensor = 50
6. turn off the motors

Give this set of commands a try.

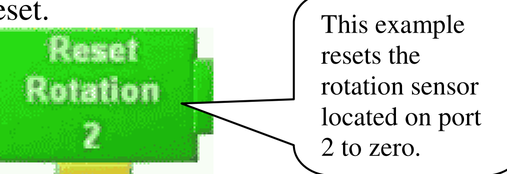
Resetting the rotation sensor

When should the rotation sensor be reset?

Again, think of driving directions to a distant place. Distances are usually given as measurements between fixed objects.

The same idea works for the rotation sensor. It is highly recommended that the rotation sensor is reset to zero whenever the robot is in a known location on the board!

Use the Reset Rotation sensor command to accomplish the reset.



Build a Fwd50 Myblock subroutine

Try putting the concepts discussed above into a Myblock routine that moves the robot 50 rotation counts.

Turning using the rotation sensor

The rotation sensor can also be used to make more accurate turns. Try making a RightTurn Myblock that uses the rotation sensor instead of the ON FOR command to make a more accurate turn.

Using variables to expand the robot's capabilities

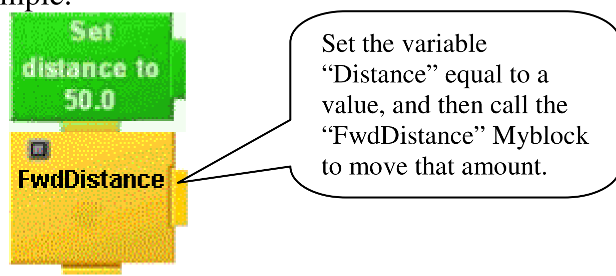
Variables are probably one of the most useful tools your kids can use to make useful functions.

Now that you have a Fwd50 Myblock command built, your robot can travel 50 rotation counts quite accurately. What if the mission requires that the robot travels 58 rotation counts? Or 125 counts?

This is where variables are useful. Try copying the Fwd50 to a new Myblock named FwdDistance. Instead of hard-coding WAIT UNTIL 50 rotations, try to set up the WAIT so it waits for a variable number of rotation counts.

Question: How will the FwdDistance command that we built know how far it should travel?
Answer: We have to tell it how far to travel before the FwdDistance command is executed.

Here is an example:



Caveat of using rotation sensor

Consider that your kids want the robot to travel until the rotation sensor reaches 100. But for some reason, the robot hits a wall at 90. What happens now?

At first, you might think that the robot would pause at the wall for a few seconds, then proceed with its program.

But think about the program – the WAIT UNTIL command is waiting until the rotation counter measures 100. Since this will never occur, the robot will wait FOREVER at the wall!!

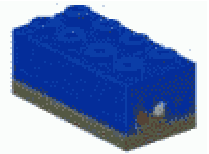
How can your kids program the robot so that the robot will not wait forever at an object if it reaches the object earlier than expected?

Topics learned in task 4:

Congratulations! You have learned more about the following topics:

- using a rotation sensor for measuring mileage during straight runs and turns
- add a rotation sensor to the robot
- Introduce variables
- When and why use variables, how to set them, how to change and read them
- enabling rotation sensor on PC
- how to make use of rotation sensor
- measuring mileage with the rotation sensor using the view mode
- how to enable rotation sensor to measure using view (must load program with rotation sensor)
- when to reset rotation sensor (every known fixed location – reset)
- comparison of accuracy using rotation sensor vs motor run time
- NEVER open a program from the programming window, always exit one screen and re-enter
  - This should avoid the need to rename myblocks (a major bug in RIS 2.0)

Programming task 5: Forward to a black line, stop, turn right



Requirements

In this lesson we will program the robot to move forward until it reaches a black line. The robot should then stop, and turn right.

Light sensor

The Mindstorms light sensor is programmed very similarly to the rotation sensor. You can view the value of the light sensor through the RCX window exactly as you did with the rotation sensor.

The light sensor interprets the amount of light from a reading of zero through 100.

Try downloading any light sensor command to the RCX, and VIEWing the light sensor reading. You have to run a program containing a light sensor to activate this feature.

Light sensor programming

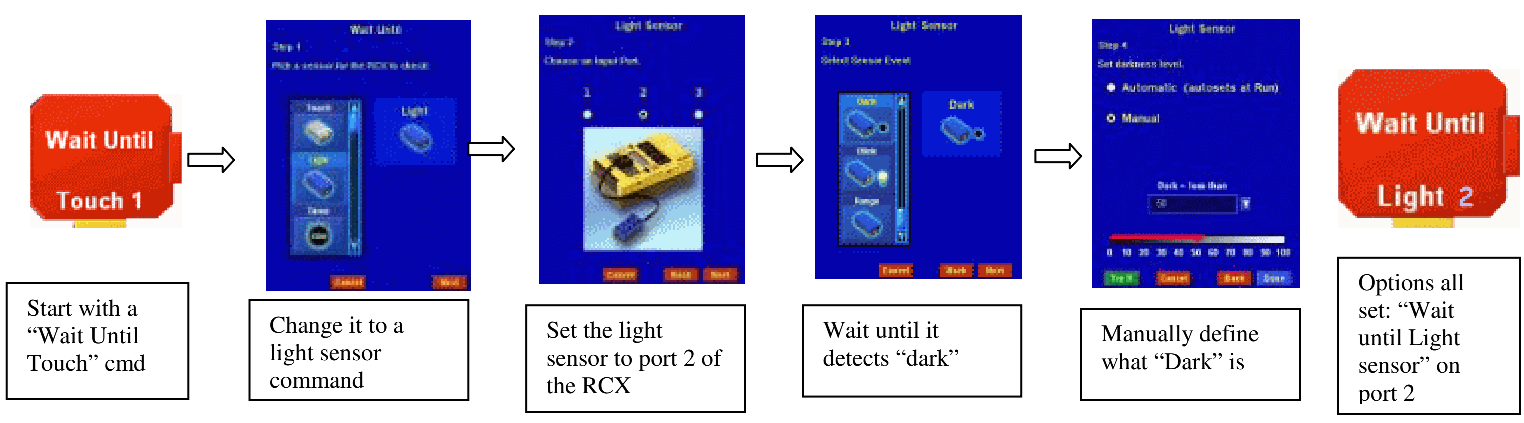
To use the light sensor in a program, we will follow the same technique that was use with the rotation sensor. Here is the psuedo-code to move forward until a black line is reached. Psuedo-code is a set of human-readable (English) statements that can easily be turned into computer commands.

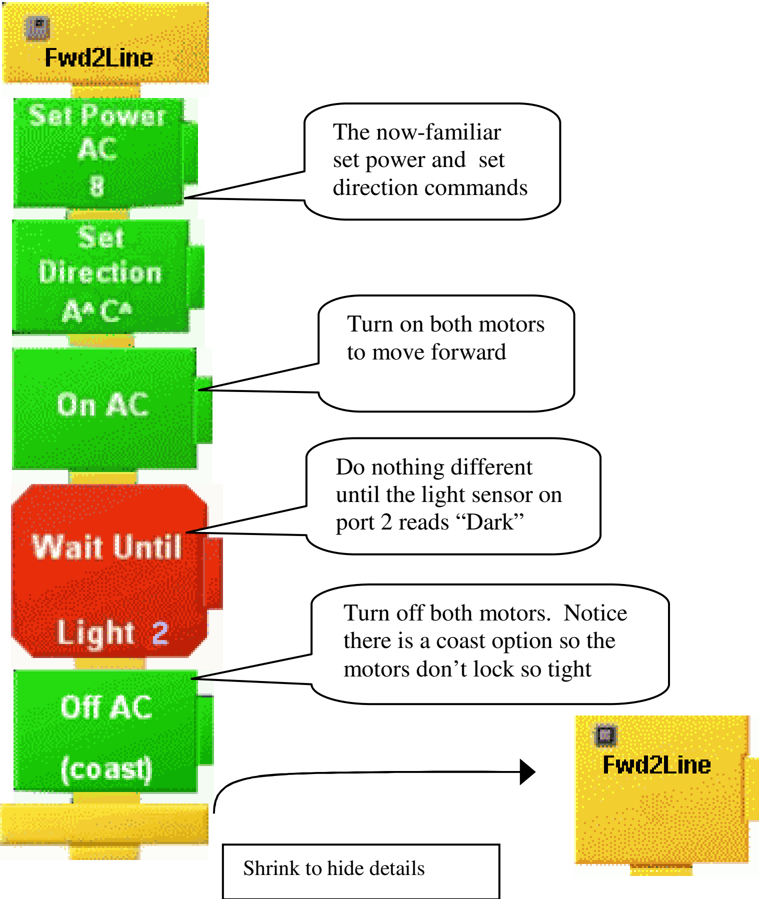
1. set motor direction
2. set motor power
3. turn on both motors
4. WAIT until the light sensor reading is dark
5. turn off the motors

Build a FWD2LINE "Myblock" command

A good place to begin is to build a FWD2LINE command. See if you can do this yourself from the above psuedo-code. Lets step through the commands.

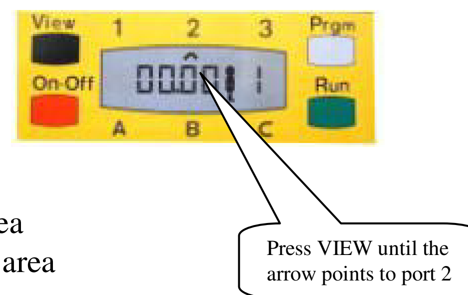
First, we need to set the proper options for a WAIT UNTIL command for the light sensor:





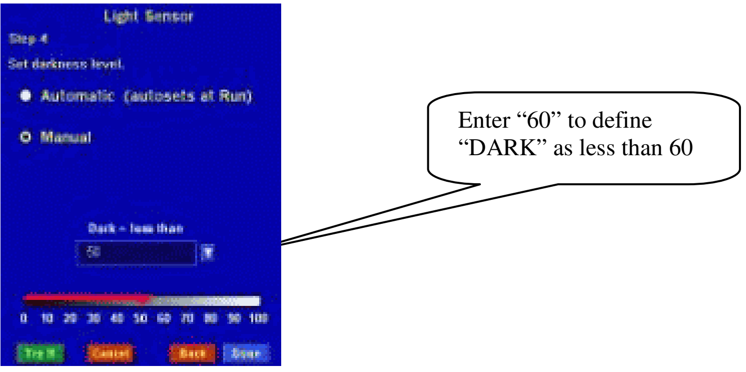
The trick here is determining what the RCX will read as "Dark". To measure with the RCX, remember that you must load AND RUN a program on the RCX which watches the light sensor.

Perform the following steps to measure the light sensor readings:

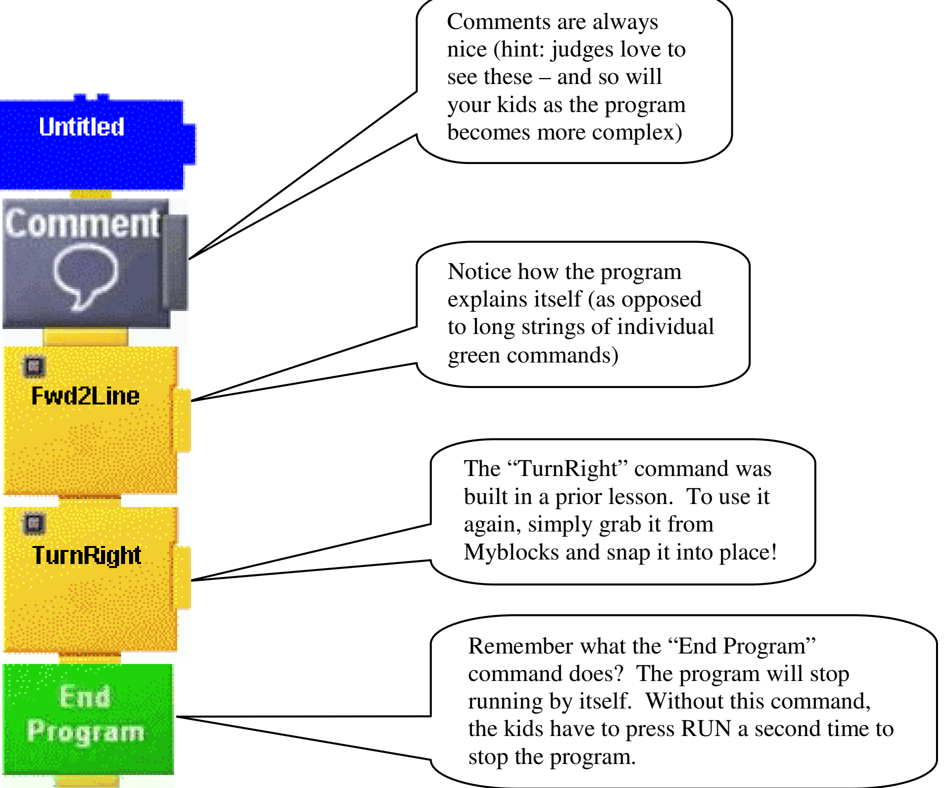


- download the Fwd2Line command to the RCX,
- run the program by pressing the green RUN button,
- stop the program by pressing RUN again
- press the VIEW button on the RCX until the arrow points to port 2.
- to determine what a LIGHT reading is, run the RCX over a white area
- to determine what a DARK reading is, run the RCX over a BLACK area
- write down the light and dark readings
- calculate the approximate average of the light and dark readings
- enter this average in the WAIT UNTIL command to differentiate light from dark.

For example, the white reading is 70, and the dark reading is 50. The average of 60 would be coded in the WAIT UNTIL command as shown:



After the black line is reached, snap a "RightTurn" myblock onto the end of the program, and you are ready to test the program.



Changing light conditions

Lets say that your kids found that 60 is the value that they want the light sensor to read as "Dark". If their program has several different light sensor commands in a program, they could "hard-code" 60 as the "dark" reading. Hard coding means to enter a specific value into a command.

If the kids take their robot to the competition and the ambient lights are much brighter, it could cause the light sensors to not see "dark" when they should, because the black line never reads below 62, for example.

Can you think of a method to address this problem?

One method is to use the automatic option on the light sensor commands. The problem with the "Automatic" light sensor reading is that it seems to be a bit unpredictable. If you start the program on a dark area, it behaves differently than if you start it on a light area.

A good method is to use a variable to store the light sensor threshold. If you want to sense "dark" as below 60, you can use the following command:

Additional notes

Which commands are used most often?

The most used commands should probably be selected from “Small Blocks” and “Myblocks”. Myblocks are the commands that your kids create themselves from small blocks.

Which commands should be avoided?

Avoid using any BigBlocks for the following reasons:

- Bigblocks are written for a specific robot – they wont work well on your kid’s robot
- Bigblocks use motor run time for measurements instead of more accurately using sensors
- Bigblocks don’t use variables to make them more flexible

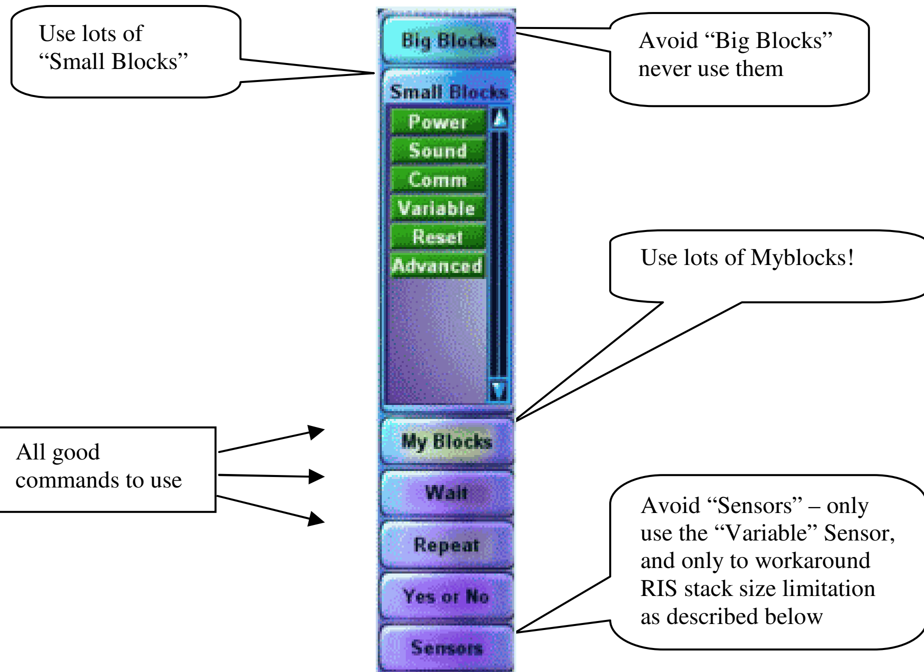
Use the blue sensor commands with care!! The blue sensor commands are sensor watchers. Once a sensor detects a value that is being watched, control jumps to a different branch of code. There seems to be no way in RIS of turning off the watcher once it is in a program.

There is a method of setting sensor watcher priorities, but this author hasn’t figured out a simple way to make this work.

To make use of sensors, best bet is to use one of the following commands to read a sensor’s value:

- WAIT UNTIL
- `Repeat Until
- Yes or NO

One exception: use the blue variable sensor on one occasion only: to get around a limitation of the RIS development environment as described below.



Commands that are not provided by RIS 2.0

RIS version 2.0 provides many commands to choose from. Sometimes your team must construct their own commands using the blocks provided!

Are there any commands missing? Sometimes your kids will want to do things that cannot be done simply with the available commands. The WAIT and REPEAT UNTIL commands only allow you to watch one sensor at a time.

Consider the following requirements:

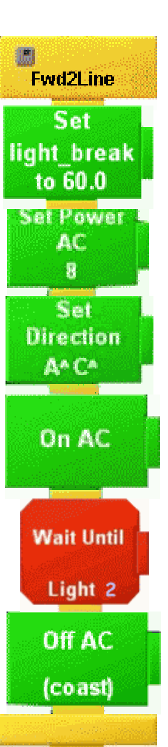
1. Move forward until the robot reaches a black line OR hits a wall (detected by a touch sensor);

2. Move the robot's arm up while traveling forward;

3. Drive forward until the left bumper AND the right bumper hits a wall.

Without giving away all of the secrets, the tasks described above can be accomplished using RIS 2.0. The trick is that more than one sensor must be monitored simultaneously.

For example, lets explore #1 above. Your kids build a program as shown:



Notice that the robot will turn on both motors to drive forward, and then WAITs until an event is detected by the light sensor on port 2.

But the requirement states that when the robot finds a black line OR if a wall is bumped, the robot must stop moving.

Will the robot stop at a black line? Yes.

Will the robot stop at a wall if it never saw a black line? NO! (Actually the robot WILL stop at a wall, of course, because it can't drive though a 2x4; but with this program, the robot will be stuck at the wall, forever spinning the wheels forward looking for that black line!)

How can the kids create a program to stop at a black line OR at a wall?

The trick is to have the kids build their own WAIT command. Think of what a WAIT command does.

- Start a REPEAT loop
- Test a sensor
- Did the sensor detect a change?
- If yes, exit REPEAT loop
- Go back to top of REPEAT loop

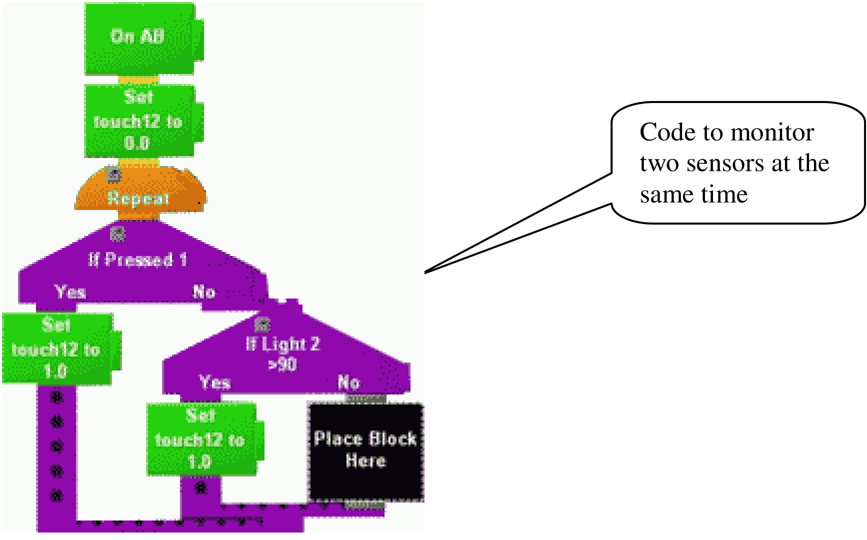
Wait for light sensor event OR touch sensor event

The program code shown below is a WAIT until touch OR light sensor event.

This portion of code is meant to give a hint at some of the more advanced programming concepts available with RIS 2.0.

There is no explanation of the code given except to say that it works.

Many FLL programming problems can be solved by constructing new commands similar to the one shown below. If your kids can understand and make use of this type of code construction, they will be ahead of the curve!



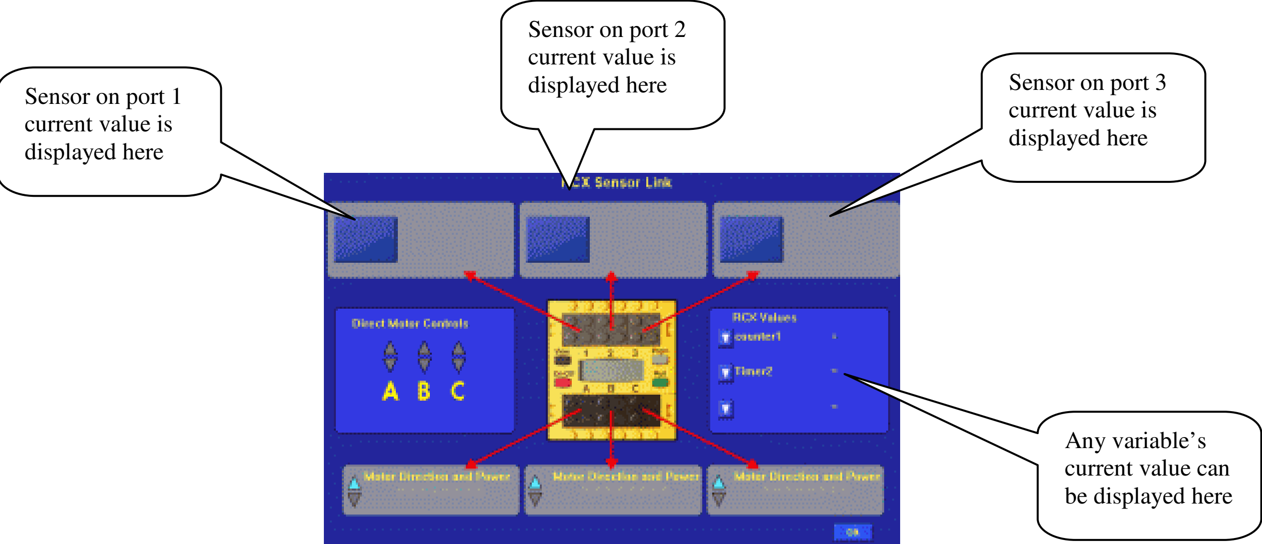
Debugging a program

One of the simplest methods of debugging is visually stepping through a program.

When this doesn’t reveal any problem areas, RIS 2.0 includes a useful debugging screen. With this screen, kids can monitor current variable values and current sensor values as the kids run their program.

This debug screen is quite sophisticated – it reads the real-time sensor and variable values from the RCX through the IR tower and displays the values on the PC screen.

To fire up the debug screen, click on the edge of the blue program title block:



If your kids still can’t figure out why the robot is behaving badly, you can try as a last resort the following:

- Isolate the problem code and run in its own program;
- Add WAIT FOR 0.1 seconds prior to sensor wait, or prior to motor on commands. This seems to allow time for the RCX to catch up sometimes;
- Reload the firmware.

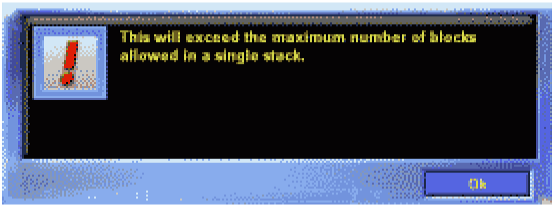
Backing up your kid’s program code

Be sure to save your team’s program after each meeting, preferably to diskette or on a different computer. The files which contain the team’s programs are very small. The program code and Myblocks are normally found in the following directory. Our recommendation is to copy the entire USERS directory to diskette.

C:\Program Files\LEGO MINDSTORMS\RIS 2.0\USERS .

Error messages:

An RIS command sequence can only be a certain length. If the number of commands exceeds the limit of several hundred commands, RIS will display the following error window :



To circumvent this problem, use the blue VARIABLE SENSOR command. The idea is to set up a second program command stack.

At the bottom of the first stack, set a variable named STACK equal to 2. This will alleviate the problem!