

Technical Learning Week 3

Review the following material on sensors. Discuss how you might use each of these sensors. When you have completed reading through this material, build a robot of your choosing that has 2 motors (connected to ports A and C), a light sensor pointed towards the ground on the front of the robot, and a touch sensor on the back of the robot. If you want and you have time, try building the bumper illustrated below. This robot will be used in the programming portion of the lesson.

Sensors provide feedback to the robot to tell it how to respond to its surroundings. The touch sensor and the light sensor perform like our own senses of touch and sight. The rotation sensor provides feedback on how many turns a wheel or gear makes. A person may program the robot to respond to each of these senses.

Touch Sensor

The touch sensor is a 2x3 gray block with a yellow button protruding from one end. A wiring harness is used to connect it to the RCX. The 2x2 plate end of the wiring harness must be placed on the top of the sensor toward the end with the yellow button. The direction that the wire protrudes from the sensor is not important. The other end of the wiring harness may be placed on one of the three numbered ports on the RCX. Once again the wire orientation is not important. Whenever the yellow button is depressed a circuit is completed and the RCX may be programmed to respond to the opening or closing of that circuit. See Figure 1 for wiring examples.

The RCX has a view function that allows one to see the action of a sensor. When the RCX is turned on, press the view button repeatedly and move the carrot (>) around the display screen until it points to the numbered port where the pressure sensor is attached. When the carrot is pointing at the port with a touch sensor, a zero will appear next to the stick figure on the display. If you press the yellow button on the touch sensor the zero(0) will change to a one(1). Touch sensors may be stacked. If a second touch sensor is wired to the same port depressing either sensors' yellow button will change the display from zero to one. Have your team members all get a chance to operate the touch sensor with the view function active.

The light sensor, rotation sensor, and motors may all be monitored using the view function on the RCX.

Light Sensor

The light sensor is a 2x4 blue block with its own wiring harness. It may be wired to any of the numbered ports on the RCX. The orientation of the 2x2 plate on the RCX port is not important. When the light sensor is active a red light shines from the end of the sensor. (When you first attach a light sensor to a port the red light may not be on. A program must be run to initiate the RCX to recognize the light sensor once it is attached to a port. Once the RCX recognizes that a light sensor is on a numbered port the red light will shine continuously, even when the program is stopped.) When the light is placed close to a surface the light reflects off of that surface and returns to the sensor. On a white or shiny surface more light reflects back to the sensor and it records a larger signal. On a black or rough finished surface less light reflects back and a smaller signal is recorded. This sensor is not a simple on/off like the touch sensor. The amount of light that returns to the sensor is recorded as a number between zero and one hundred. Wire the light sensor to port 2 on the RCX and use the view function to see the output of the sensor. The display will show a number between zero and one hundred. Point the sensor at various surfaces and see how the value changes. Have your team members perform this test.

Bright lights in room increase the light returning to the sensor and it records a larger signal. This may be tested by turning the room lights on and off while viewing the output of the sensor using the view function. Notice that most of the readings vary between 20 and 60. It is difficult to find a surface that produces a reading near zero or one hundred. If a light sensor is used to follow a dark line the view function may be used to insure that the numeric value returned to the RCX for that dark line versus value returned for the white background crosses a threshold value that you set in the software.

Rotation sensor

The rotation sensor is a brick with a rotating shaft extending from it. When wired to the RCX it counts each $1/16^{\text{th}}$ of a rotation as one unit. A full rotation of the sensor shaft returns a value of sixteen. A series of gears may be used to connect the rotation sensor to a robot wheel, or other rotating part on the robot to be sensed. A gear train may increase the number of rotations made by the sensor shaft versus the point of the robot being monitored to get higher precision in the rotation sensor. Think back to the extreme gear example to understand how this is accomplished. The view function may be used to see how the rotation sensor counts as the shaft turns.

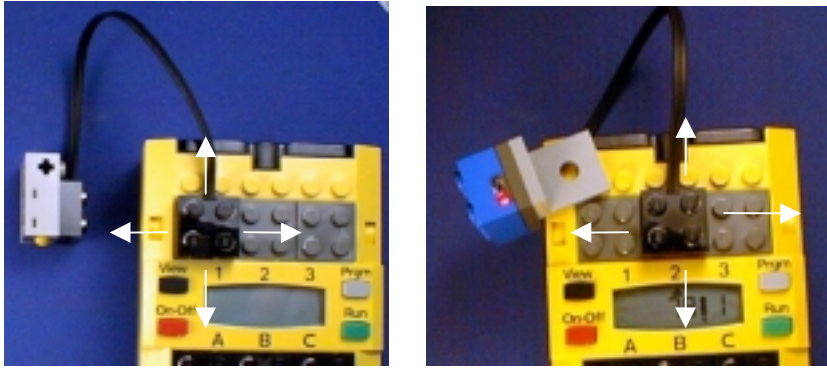
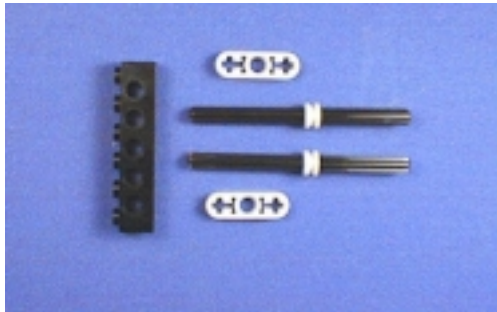
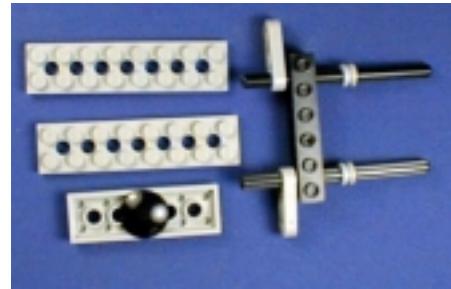


Figure 1. Touch and Light sensors may be oriented in any direction.

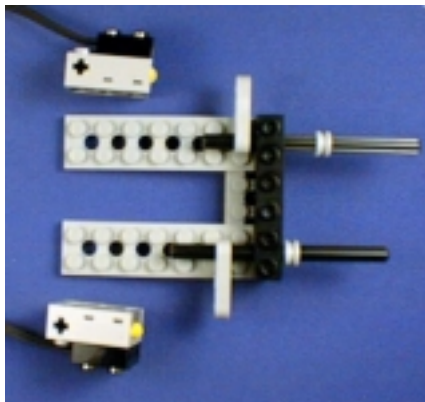
Here is one way to build two touch sensors into a robot.



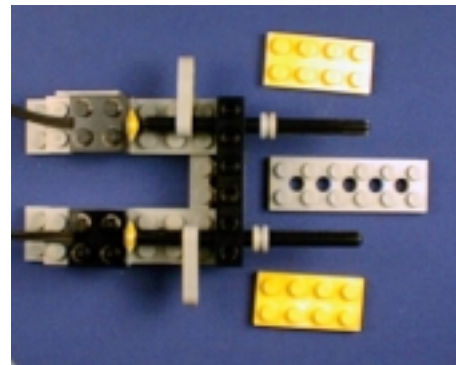
Step 1 Find two axles, a beam and the four smaller parts.



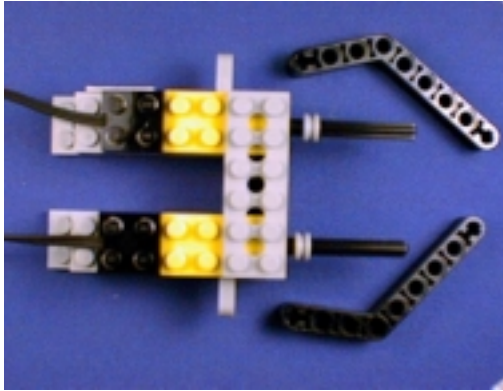
Step 2 – The axles slide in the holes in the beam.



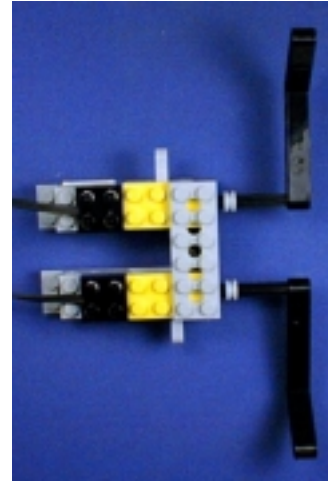
Step 3 – add the touch sensors



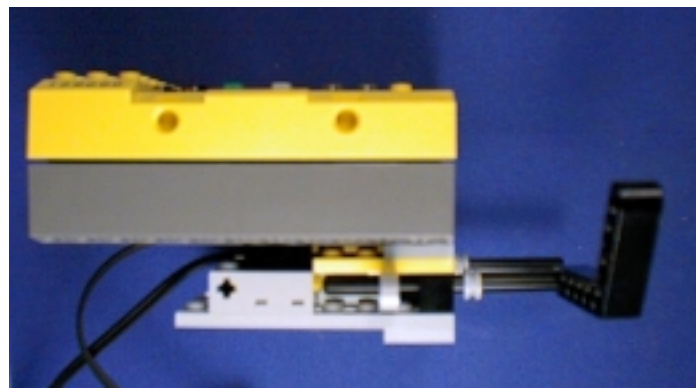
Step 4 – add these plates to complete the frame for mounting to the RCX.



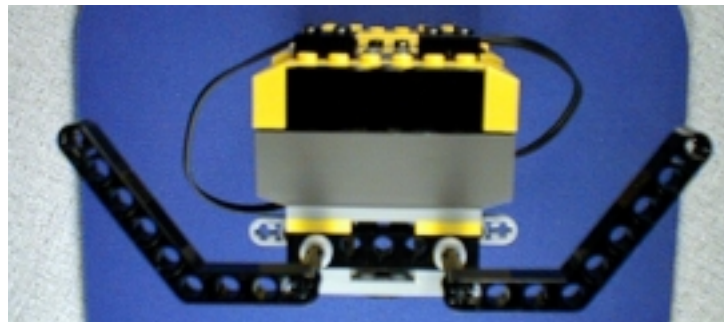
Step 5 – attach the “bumpers”



Step 6 – The finished sub-assembly.



The bumpers attached to the RCX.



And a front view. Add wheels and motors and your robot is ready to find its way around.

Introduction to Robolab programming

The lesson plan for week 1 included an overview of the RoboLab programming environment. A RoboLab program is a sequence of instructions that controls how the robot responds to things it encounters in its environment. The robot gets information about its surroundings through sensors. It then makes decisions about how it is going to move. It senses, and reacts, just like we do. Programs are built wiring together command boxes. Each command box represents a single instruction, and commands are executed in the order that they are wired, forming a sequence of instructions for the robot to follow. An overview of all the commands available for use in RoboLab can be found in the documentation. In addition the program has built in context help that can give you information on individual commands. In order to activate this function, enter the programming mode and select show context help under the help menu.

Sensors provide the RCX with information about the outside world. A light sensor provides the RCX with a number that indicates the intensity of light. Touch sensors inform the RCX whether they are 'pressed' or 'released', telling the robot when it runs into things. Timers provide the RCX with a number that indicates the amount of time that has passed. Rotation sensors provide a number that is a count of complete rotations of an axle. Programs can read these sensor values in order to make decisions about what to do.

Below is a brief description of the most important RoboLab commands that control how a robot behaves. This is only a brief description of a small selection of commands available. Many of the commands listed below are selected to be examples of a larger set of similar commands.

Output Commands



Motor A/B/C Forward

This command will turn on motor A, B, or C depending on the command used in the forward direction. In addition, a speed modifier can be added to this command to control how fast the motor will turn. If there is no speed modifier, the speed will be set to full.



Motor A/B/C Backwards

This command will turn on motor A, B, or C depending on the command used in the backward direction. In addition, a speed modifier can be added to this command to control how fast the motor will turn. If there is no speed modifier, the speed will be set to full.



Motor Forward

This command will activate the outputs specified by modifiers in the forward direction. If there are no output modifiers, then all outputs will be turned on. In addition, a speed modifier can be added to this command to control how fast the motor will turn. If there is no speed modifier, the speed will be set to full.



Motor Backwards

This command will activate the outputs specified by modifiers in the backward direction. If there are no output modifiers, then all outputs will be turned on. In addition, a speed modifier can be added to this command to control how fast the motor will turn. If there is no speed modifier, the speed will be set to full.



Stop A/B/C

This command will stop the either output A, B, or C depending on which command is used.



Stop ABC

Any output from the RCX on any of its ports will be stopped by this command.



Stop

This will stop the outputs of any output specified by attached modifiers. If there are no modifiers attached, then all outputs will be stopped.

Wait for Commands



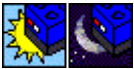
Wait for Time

This command will wait for the time specified in the attached numeric constant modifier in minutes. If there is no attached numeric constant then it will wait for a default of 1 minute. In addition to this command there are several other commands about waiting for predefined times.



Wait for Push/Letgo

These commands will wait for the button to either be pressed or released depending on the command. The sensor that should be monitored is defined by attaching sensor port modifiers. If no sensor port is specified, then the default is sensor port 1.



Wait for Light/Dark

These commands will wait for the light sensor to either be above or below a threshold depending on the command. The sensor that should be monitored is defined by attaching sensor port modifiers. If no sensor port is specified, then the default is sensor port 1. A numeric constant can be attached to define what number it must be above or below to continue. The default value if there is not an attached constant is 55.



Wait for Brighter/Darker

These commands will wait for the light sensor to either be increase or decrease by a specified amount depending on the command. The sensor that should be monitored is defined by attaching sensor port modifiers. If no sensor port is specified, then the default is sensor port 1. A numeric constant can be attached to define how large a change needs to be before the program continues. If no numeric constant is attached, then the change has to be 5 units.

Structures



Touch Fork

This command makes the program run along one of two paths depending on the if the touch sensor is pressed or released. A sensor port modifier can be attached to this command to define which port the command should monitor for the touch sensor. Both paths that come from this command must be recombined with a fork merge command.



Light Fork

This will split the path depending on the value of the light sensor compared to some standard value. A sensor port modifier can be attached to define which port the command should monitor for the light sensor. If no modifier is attached, the default port is 1. A numeric constant can also be attached to set the value that the sensor should compared to. If no numeric constant is attached, the default value is 55. Both paths from this command must be recombined with a fork merge command.



Light sensor Equal Fork

This will split the path depending on if the value of the light sensor equals some value. A sensor port modifier can be attached to define which port the command should monitor for the light sensor. If no modifier is attached, the default port is 1. A numeric constant can also be attached to set the value that the sensor should compared to. If no numeric constant is attached, the default value is 55. Both paths from this command must be recombined with a fork merge command.



Fork Merge

This command combines the separate paths from a fork command back into a single path. A fork merge command is required for every fork.



Red/Blue/Yellow/Green/Black Land

This command matches with the same colored jump command. It is the landing point for the jump command.



Red/Blue/Yellow/Green/Black Jump

This command will jump from the jump command and continue running at the same colored land command.



Start of Loop

This command begins a loop that runs all of the commands between the start of loop and end of loop. The number of times that the loop executes depends on the value of an attached numeric constant. If no numeric constant is attached then the default number of loops are 2.



End of Loop

An end of loop command will return to the start of loop command. Every start of loop command requires an end of loop command to contain the commands that will be looped.



Loop while Touch Sensor is Pushed

This command replaces and functions like the start of loop command except that instead of looping a specified number of times, it loops as long as the touch sensor is pushed. A sensor modifier can be attached to define which port to monitor for the touch sensor. If there is no modifier attached, it defaults to sensor port 1.



Loop while Touch Sensor is Released

This command replaces and functions like the start of loop command except that instead of looping a specified number of times, it loops as long as the touch sensor is released. A sensor modifier can be attached to define which port to monitor for the touch sensor. If there is no modifier attached, it defaults to sensor port 1.



Loop while Light Sensor Is Less Than

This command replaces and functions like the start of loop command except that instead of looping a specified number of times, it loops as long as the light sensor is less than a specified value. A sensor modifier can be attached to define which port to monitor for the light sensor. If there is no modifier attached, it defaults to sensor port 1. A numeric constant can be attached to set the value that the light sensor is compared against. If no numeric constant is attached, the value is set to 55.



Loop while Light Sensor Is Greater Than

This command replaces and functions like the start of loop command except that instead of looping a specified number of times, it loops as long as the light sensor is greater than a specified value. A sensor modifier can be attached to define which port to monitor for the light sensor. If there is no modifier attached, it defaults to sensor port 1. A numeric constant can be attached to set the value that the light sensor is compared against. If no numeric constant is attached, the value is set to 55.

Modifiers



Output A/B/C

This modifier affects most commands that output from the RCX. These modifiers define on which output the command will act.



Input 1/2/3

This modifier can be used on most commands that accept inputs from the RCX. It defines which input the command will read.



Power Level 1-5

These modifiers affect motor output commands to define the strength that the motor will run.

123 Numeric Constant

Numeric constants are modifiers that can be attached many commands. They are used to modify any command that operates on or with numbers.

Write a program that turns the robot in place

You may want to have the lesson from week 1 handy to help you through these steps.

A robot is turned by giving different commands to the left and right motors. Motors can be driven in different directions, at different power (speed of rotation), and for different amounts of time (controlling the duration of the turn and the ground traveled). If the left motor rotates forward and the right motor backward at the same speed the robot will turn in place. If both motors rotate in the forward direction, but at different rates, the robot will move through an arc.

This program will look a lot like the one in lesson 1, described in 'Add a turn to your program', which is illustrated in Figure 3 of that lesson.

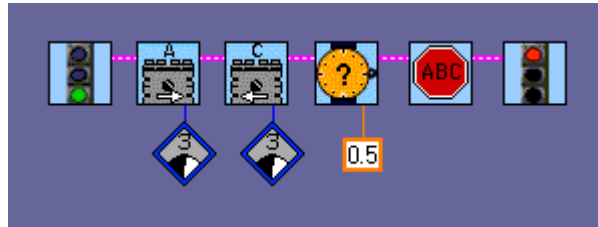
1. Plug one motor into port A and the other into port C.
2. Begin by wiring **Motor commands** so that motors A and C move in opposite directions.
3. Wire on **Motor speed Modifiers** to set the speeds for the motors. Begin with the same speed for both motors.
4. Next, turn the motors on for a specified time. Add a **Wait for Time** command and wire on a numeric constant that contains the value in seconds that you think will turn the robot 180°.
5. Add a **Stop All Outputs** command and wire the final command to the red light.
6. Download the program and execute it. Does it work as you expected?
7. If the turn is not correct, adjust the time appropriately. Download the program and notice how the turn changed. Repeat the process until the desired turn is achieved.
8. The combination of power and time controls the turn. Once one combination of values that works is discovered, change the power level and repeat steps 4–6 until it works correctly.

One thing to note is that these values can change depending on how new the batteries are. You'll probably have to experiment to find good values for the different stages of battery life.

☺ **Congratulations!!** ☺

Now you can turn the robot on a dime when space is tight.

Your program should look similar to this. The specific values will differ significantly based upon differences in robot design, battery state, and chosen motor speed.



Write a program that moves the robot through an arc

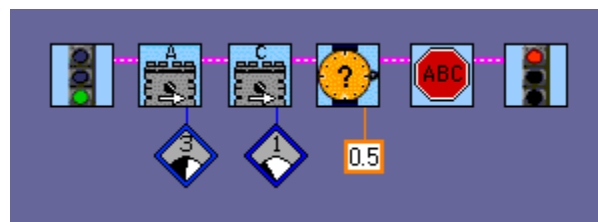
When a car turns, it doesn't turn in place, but actually moves through a curve (arc). That means it changes its direction (heading) a little at a time while also moving forward (or backward).

1. Wire commands to drive motors A and C to move in the same direction using a different motor command for each motor.
2. Attach a different motor speed command to each motor command. Can you predict which way the robot will turn?
3. Next, use the **Wait For Time** command to turn on the motors for a specified amount of time of your choice.
4. Add a **Stop all Outputs** command after the wait for time command.
5. Down load and execute the program. Does it work as you expected?
6. Adjust the timings to change the turn to attempt to make the turn closer to the desired turn. Download, execute, and modify the program until the desired turn is made.

☺ **Congratulations!!** ☺

Now you can turn the robot in an arc.

Your program should look similar to this. The specific values will differ significantly based upon differences in robot design, battery state, and chosen motor speed.



Use test panel to get sensor values for 'dark' and 'light'

1. Attach a light sensor to input #1 on the RCX brick.
2. Open the test panel by selecting the **Project** menu and then selecting **Interrogate RCX**. Ensure at this time that the RCX is on and pointed towards the IR tower.
3. Click on the gray box near to the #1. This should open up a new small gray box which contains two text boxes and a numeric output. Set the top text box to reflection by clicking and selecting that choice in the list. Set the lower box to percent by a similar method.
4. Point the light sensor towards the light part of the test area. Ensure that during this entire process that the RCX stays in range of the IR tower. Record the number that is shown on the computer's screen. Move around the light source and record how that changes the sensor values.
5. Point the light sensor towards the dark part of the test area. Record the number that is shown on the computer's screen. Move around the light source and record how that changes the sensor values.
6. Determine a cutoff value half way between the average light and dark readings.
7. When you are done, click back to return to the programming area.

You now know how to interpret sensor readings. Readings above the cutoff value can be interpreted to indicate a light surface, readings below this value a dark surface.

☺ **Congratulations!!** ☺

You now understand that the numerical sensor reading changes when different surfaces are 'read', and also when the amount of light is changed.

Write a program that moves the forward until it moves across a black line and the convert this program into a subVI.

This example has two objectives. The first is a set of program commands that will move the robot forward until it crosses a black line and then stops. The second objective is to convert this set of commands into a subVI. You can imagine circumstances in which it would be useful to have the robot move forward until it crosses a black line. When a subVI has been created to do this task, it can be reused without having to recreate the code again. You will need a robot that has a light sensor on the front of the robot pointing toward the ground attached to port 1.

1. Start the program by selecting the **Motor Forward** command and attaching the proper modifiers to move the robot forward.

When attempting to choose the next command to use, form a sentence to explain what you want the robot to do. This sentence will often help to indicate which command you should use. For this program we want the robot to move forward while it waits for it to reach a black line. The words *wait for* indicate that a **Wait for** command would be appropriate for the situation. Another example would be 'If the counter indicates that 3 lines have been crossed then turn right, otherwise go straight.'. Whenever you use *if then*, this is an indication that a **Fork** structure should be used.

2. Add a **Wait for Dark** command after the motor forward command. Add to the wait for dark command an input #1 modifier.
3. Add a numeric constant modifier that contains the value that you found by using the interrogate RCX screen. The program will wait until the light sensor falls below that value and then the program will

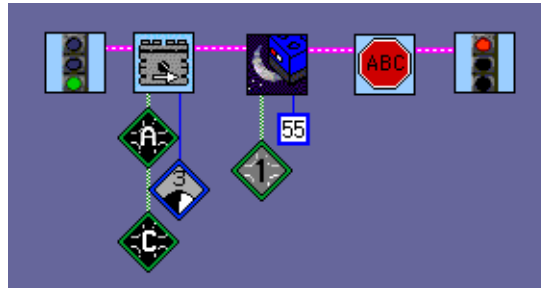
continue executing the code. Because the motors were on when the command began waiting, the robot will continue to go forward while the command is waiting.

4. Add a **stop all outputs** command after the wait for dark command.

😊 **Congratulations!!** 😊

You have created a program to move the robot forward until it crosses a black line.

Your program should have looked similar to this before it was converted into a subVI. The specific values will differ significantly based upon differences in robot design, and battery state.

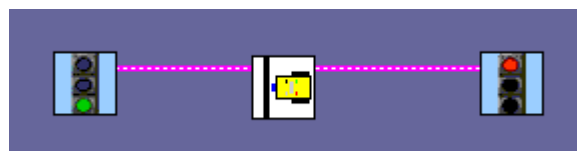


Make this set of commands into a subVI

1. Select all of the commands in between the green and red lights by clicking and dragging a box around them with the select tool.
2. Click on the **edit** menu and then select **Create SubVI**. This command will take all of the commands and turn them into a single command called a subVI.
3. Double click on the newly created subVI to open it in a new window. In the upper right hand corner of the window there is an icon. By double clicking on that icon, you can edit it so that your subVI is easily recognizable.
4. Once you are done editing the icon save the subVI like you would any other program making sure to remember its name, something like **toline**, and location, probably just the default. Close the window that contains the subVI.
5. Whenever you need to load that subVI, click on the **Select a VI** command in the main menu and select the subVI that you had saved, however, that is not necessary at this time.
6. Download and test the program.
7. If the program need adjustment, double click on the subVI to open it up in a new window. Go to the **window** menu and select **Show Diagram**. Now you can edit this program like any other program. When you are done editing the program, save and close the window.

Your new subVI can be used just like any other command

Here is how it looked after I converted it to a subVI



By packaging code into these **subVI's**, you make programs easier to read, write, and debug. Imagine if you have multiple programs that all have to go forward until a black line is crossed. Having the **toline** command makes it easier to add this function to each of the programs and makes it easier if the lighting conditions change. You simply have to change the value in one location instead of in each program. Additionally, when you start building big programs, you will find by referring to a set of commands with a single block makes the overall program much easier to read and to understand. *Code modularity* refers to this packaging of stacks of code into a single command (sometimes referred to as a *subroutine*).

Write a program that moves the robot backward until its touch sensor is hit and then convert this program into a subVI

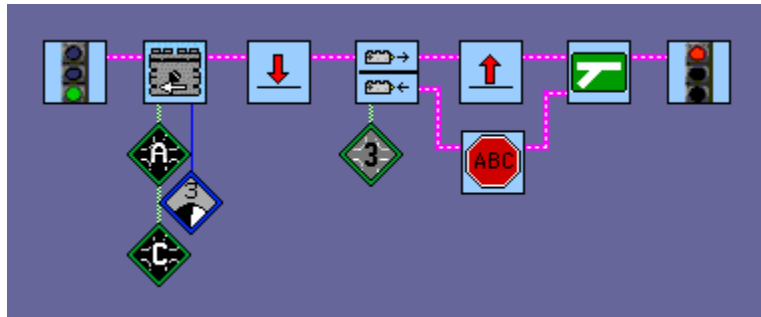
Similar to the last example, we could use a **wait for** command to write this program. However, in this example we will use the conditional fork. In this situation you might prefer to use the **wait for** block because it would be simpler, but we will use the conditional fork here to learn how they work. It is often the case in programming that there are multiple ways to achieve the desired behavior. When trying to decide which way to program, think about what will be the easiest to read, to write, and to debug. Simplicity is key when programming, however we will temporarily ignore that rule in order to introduce several more commands.

1. First clear the programming area so we can begin with a blank program.
2. Add a **Motor Reverse** command and attach the necessary modifiers to make motors A and C go backwards at the speed of your choice.
3. Select the **Structures Menu** and then select the **Jumps Menu**. Add after the motor command a **Red Land** command. This command will be returned to whenever a **Red Jump** command is executed.
4. Return to the structures menu and enter the **Forks Menu**. Add after the red land a **Touch Sensor Fork**. What this command does is it splits the direction that the program will follow depending on the state of the touch sensor. Add the correct modifier to the fork so that it is monitoring the correct port.
5. Wire to the top, which is the direction it takes if the touch sensor is not pressed, a **Red Jump** command. This will cause the program to loop until the touch sensor is pressed.
6. Attach a **Stop All Outputs** command to the lower part of the fork so that when the button is pressed the robot will stop.
7. Any fork that is created will also have to be recombined. Enter the forks menu and add a **Fork Merge** command. Wire the output of the stop all outputs command to one of the inputs of the merge command and wire the other input to the output of the red jump command. Wire the fork to the red light.
8. Download and test your program. Assuming that it works, convert it to a subVI and save it.

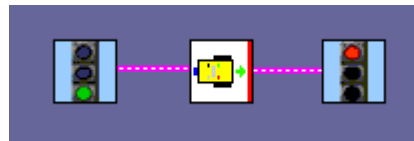
☺ **Congratulations!!** ☺

You have created a program to move the robot backward until its touch sensor is bumped.

Your program should have looked similar to this before it was converted into a subVI. The specific values will differ significantly based upon differences in robot design, and battery state.



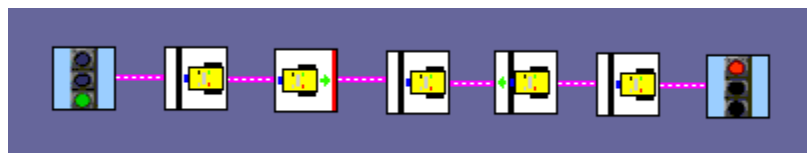
Here is how it looked after I converted it to a subVI



Now, let's put it all together! Write a program that moves the robot forward until it crosses a black line, then moves backward until it's bumper sensor is hit, then moves forward until it crosses 2 black lines.

1. Clear the programming area.
2. Click the **Select VI** command and choose the command that you made to move the robot to a black line.
3. Either program a new more efficient method to go backwards until the touch sensor is touched, or use the subVI that you created previously.
4. Add the commands necessary to move the robot to the first black line again.
5. Add the commands necessary to move the robot over the black line to the area after it. Hint: this is the same as moving to a black line except you start on black and move to white.
6. Add the commands necessary to move the robot to the second black line.
7. Download and test your program. Save it if you like. (To test it, you can use the test pad or use black tape on the floor.) If the program operates correctly you may wish to clean up the code by creating subVIs for the tasks that are not yet subVIs.

The figure below is a sample program that performs the task as explained above. Notice that only subVIs were used. This makes reading the code much easier.



☺ Congratulations!! ☺

You have completed this lesson. This lesson covered a lot of material. You did a great job of getting through all of it!
I hope you had fun.