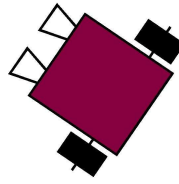


Lab book

Exploring Robotics

(CORC 3303)

Dept of Computer and Information Science
Brooklyn College of the City University of New York
updated: Fall 2011 / Professor Elizabeth Sklar



Brooklyn College
The City University of New York

Exploring Robotics (CORC 3303)

©2011

UNIT A Lab, part 1 : Robot Construction and Gearing

vocabulary

- RCX
- gear
- motor
- RoboLab
- communication tower
- sensor
- touch sensor
- light sensor
- connecting lead
- axle
- beam
- brick
- bushing
- hub
- peg
- plate
- pulley
- spur
- tire

materials

Make sure you have all of the following materials before you start the lab:

- lego robot kit
- instructions for assembling the robot

instructions

1. assemble the robot

- Follow the building instructions and construct the robot. Some of the parts you will use include:



RCX



motor



gear



axle



touch sensor



connecting lead



light sensor



beam



1x4 plate



2x8 plate



bushing



half bushing



peg



hub



pulley



tire

2. download the test program

- The RCX is a **microprocessor**, a small computer which controls what the robot does. The RCX needs a **program** that contains **instructions** for controlling the robot. We will use an application called **RoboLab** to program your robot. RoboLab runs on a PC or a Mac and includes an environment in which you can compose programs for the robot. A program must be **downloaded** — transferred from the PC or Mac to the RCX.

For today, your instructor will write a **test program** using RoboLab and download it for you; but later, you will learn how to write your own programs in RoboLab.

- The **communication tower** looks like this:



It is the device that facilitates downloading. It is connected to the computer's USB port. The robot is placed next to the tower, and the RoboLab user clicks on the **download arrow**



to send the program to the robot.

When the download is finished, then the RCX will sing a happy tune :-)

3. run the test program

- After the program downloads successfully, put your robot on the floor and turn it on by pressing the **ON** button.
- Then press **RUN**.
- What did your robot do? Write your answer below. Be precise!

4. modify the robot

- You can use **gears** to control the robot's speed — the way in which power is transferred to the robot's wheels from the motor. This is called using **gear ratios**. Your robot kit has several gears, each of different sizes. The size is measured by the number of **teeth** on the gear. How many different gears do you have and what sizes are they?

- When two gears are meshed together, if they have the same number of teeth, then every time one gear goes around, the other does too:



8-tooth gear 8-tooth gear

However, if one gear is larger than another, then it will rotate more slowly than the smaller gear:



24-tooth gear



8-tooth gear

In the example above, how many times does the 8-tooth gear rotate for each rotation of the 24-tooth gear?

-
- Draw below the current configuration of gears used in your robot. Be sure to indicate which gear is connected to the **motor** and which gear is connected to the **wheel**.



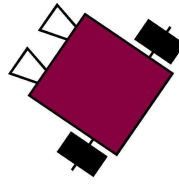
- Using gears to make your robot go slower is called **gearing down**. Using gears to make your robot go faster is called **gearing up**. Is the current configuration of your robot gearing up or down?

-
- Modify your robot to use a different gearing configuration. Draw the new configuration below, again remembering to indicate where the motor and wheel are connected.



- Is the new configuration of your robot gearing up or down?

-
- Now run the test program again. Does the robot go further or not as far as it did before? Why?



Brooklyn College
The City University of New York

Exploring Robotics (CORC 3303)

©2010

UNIT A Lab, part 2 : Introduction to RoboLab

vocabulary

- RoboLab
- communication tower
- canvas
- icon
- drag-and-drop
- function palette
- tools palette
- program
- algorithm
- syntax error
- logical error
- wire
- modifier
- numeric constant
- random number

materials

Make sure you have all of the following materials before you start the lab:

- lego robot kit
- communications tower

instructions

1. start up RoboLab

- Find the **RoboLab** icon on your computer and double-click on it to start it up.


















- Click on **PROGRAMMER**. Double-click on **INVENTOR 4**. Your screen will look like this:



- Now you are ready to start programing in **RoboLab**!

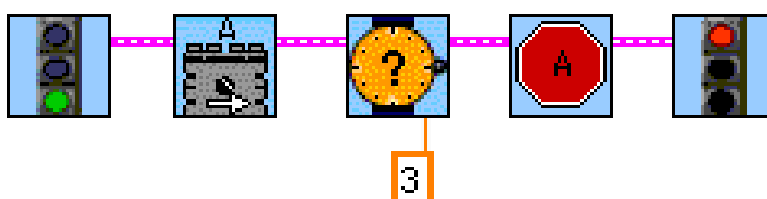
2. find the icons

- Locate the **functions palette**. See if you can find each of the following **icons**. Make a note of which sub-palette each icon belongs to. As you learn what each icon does, write down its function in the boxes below.

3. create your first program

- Use the mouse to select, drag and drop RoboLab **icons** onto your canvas and **wire** them together, creating the following program:



- What do you think it does? Write down your ideas below:
-
-

4. download your program



- Find the **communication tower**, which looks like this:
- Connect it to your computer's USB port (ask if you need help finding it).
- Place your robot next to the tower.



- Find the **download arrow** in your RoboLab window that looks like this:
It's up near the top, just under "File Edit" menu options. Click on it, and your program will **download** onto your robot!
- If the download works, then the RCX will sing a happy tune :-)
- If you have trouble, check these things:
 1. The RCX must be turned on during the download.
 2. The black part of the RCX should be turned towards your tower.
 3. If there is a lot of light in the room, try covering your RCX and the tower with a piece of paper.
 4. If your download arrow is broken, then you have an error in your program. Make sure that you copied all the icons correctly and that the wires all start and end in the right places.
 5. Make sure the **touch sensor** is connected to **port 1**.

5. run the program

- After the program downloads successfully, put your robot on the floor and turn it on.
 - Then press **RUN**.
 - What did your robot do? Is that what you expected? Write your answer below.
-
-

6. modify the program

- Now change the program to make your robot go in a straight line for 5 seconds and then stop. **Edit** the program in RoboLab, **download** it again and **test** it. Repeat this process until it works!
- Then in the box below, draw your code:



7. programming challenges

- Complete as many of the following programs as you can.
- After you get each program to work, draw the code in the boxes provided on the next page.

(a) Program the robot to go forward for 2 seconds and then go backward for 2 seconds and then stop. Is it back where it started from?



(b) Program the robot to go in a square.


i. Design your code before you start programming. Write your *algorithm* below.


ii. Now in the box below, draw your **code**




8. random numbers


- The next challenges use the concept of **random numbers**. There are two new icons that you will


need for these challenges: the *wait for random time* icon, which looks like this:  and the

motor random icon, which looks like this: .

-  **Here is how the *wait for random time* icon works:**

The icon tells the robot to wait for a random amount of time. The default is to wait for a random

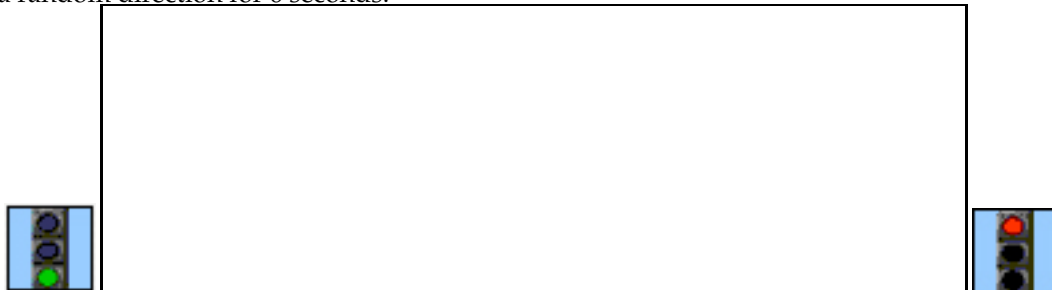
amount of time between 0 and 5 seconds. You can use the **numeric constant** modifier  to set the maximum random time (in seconds). For instance, we can set the maximum random time to 10 seconds in order to pick a random time between 0 and 10 seconds. You should set the modifier to the maximum random number you want to wait for.

-  **Here is how the *motor random* icon works:**

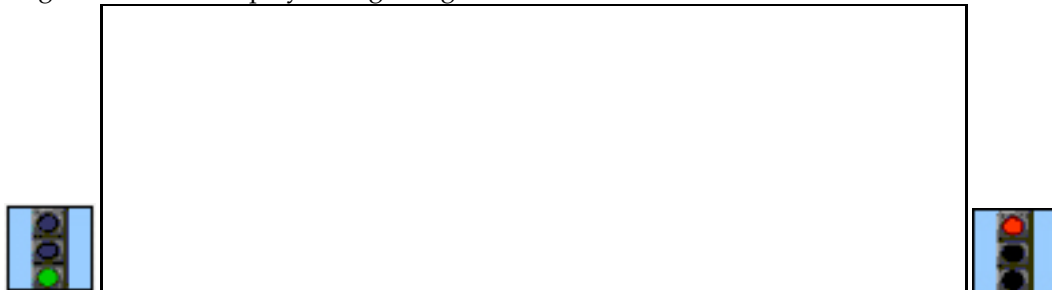
The icon turns on motors to run in a randomly chosen direction (either forward or in reverse). You can use the port modifier to choose which motors to turn on by stringing together any combination of output port modifiers A, B and C. You can also choose the power level by stringing a single modifier indicating the power level for the motors.

9. programming challenges

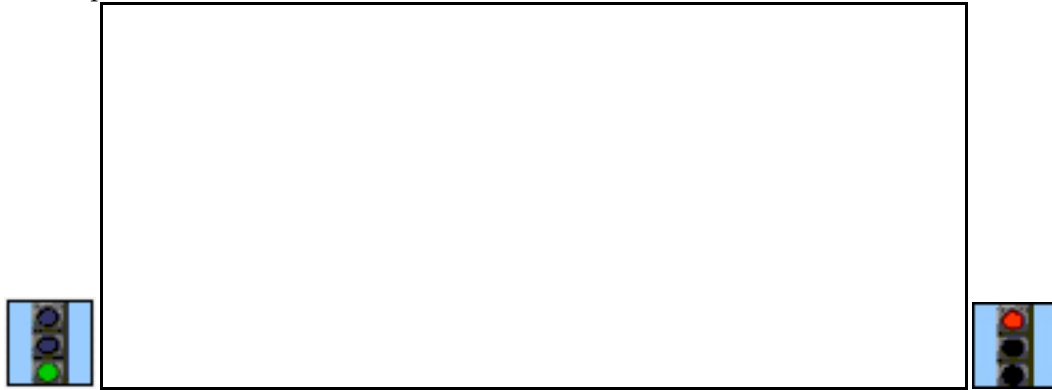
- Complete as many of the following programs as you can.
 - After you get each program to work, draw the code in the boxes provided on the next page.
- (a) Program the robot to go forward for a **random** number of seconds (between 0 and 7), and go in a random direction for 6 seconds.

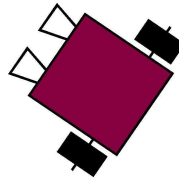


- (b) Program the robot to play a song using the music notes.



- (c) Program the robot to go forward for 2 seconds using power level 1 while playing note A; then go forward for 2 seconds using power level 2 while playing note B; then go forward for 2 seconds using power level 3 while playing note C; then go forward for 2 seconds using power level 4 while playing note D; then go forward for 2 seconds using power level 5 while playing note E; then stop.





Brooklyn The City
College University
of New York

Exploring Robotics (CORC 3303)

©2010

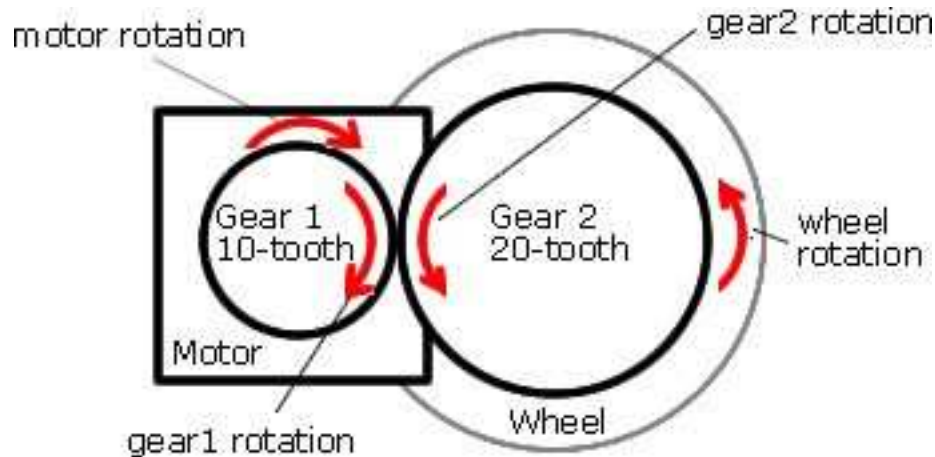
UNIT B Lab, part 1 : Robot Gears and Program Control

vocabulary

- gear ratio
- rotational speed
- revolutions per minute (rpm)
- angular velocity
- torque
- idler gear
- robot control

gears

In our previous lab sessions, we have used gears for reversing the direction and changing the speed of rotation of the wheel. Consider the gear configuration below:

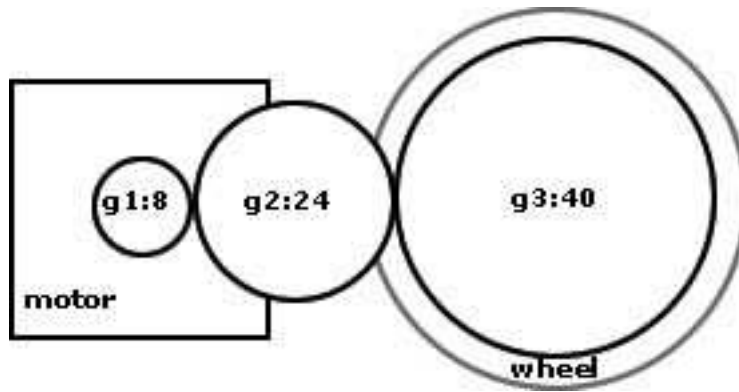


In the image above, there are 2 gears: 10-tooth and 20-tooth. Gear 1 is connected to a motor, and Gear 2 is connected to a wheel. The gear connected to the motor is called the *drive* gear. Assume the motor is spinning clockwise at 100 rpm (revolutions per minute). How many times does the wheel rotate in a minute and in which direction (clockwise or counterclockwise)?

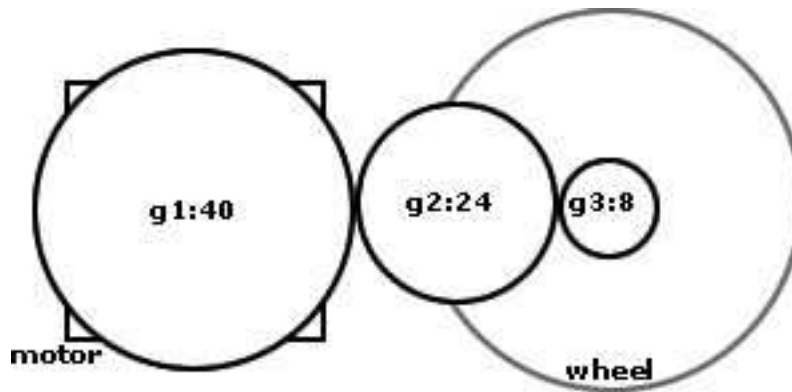
Answer:

- i. For each revolution of the motor, the drive gear (Gear 1, 10-tooth) will rotate once in the same direction as the motor (i.e., clockwise in this case).
- ii. In each revolution, Gear 1 will touch 10 of the teeth in Gear 2 (each tooth in Gear 1 will interlock with a tooth in Gear 2 and force Gear 2 to turn). Thus, Gear 2 will turn half a rotation in the opposite direction (i.e., counterclockwise in this case). The *gear ratio* is defined as the ratio of the gear being driven (in this case, Gear 2) to the gear doing the driving (in this case, Gear 1). In our example, this ratio is "Gear 2" to "Gear 1", or 2 : 1 (read "two to one"), meaning that for every two revolutions of the drive gear (Gear 1), the interlocking gear (Gear 2) will turn one revolution. This can also be stated as saying that for one rotation of Gear 1, Gear 2 will complete $1/2$ a rotation.
- iii. Since Gear 2 is attached to the wheel, the wheel will also turn $1/2$ a rotation for each rotation of the motor, counterclockwise in this case.
- iv. The motor spins 100 revolutions per minute. How many revolutions does the wheel rotate in a minute? We can find the answer by multiplying the number of revolutions the motor completes in a minute by the wheel revolution rate that we found above (i.e., $1/2$). Therefore the answer is : $1/2 * 100 = 50$ revolutions, counterclockwise.

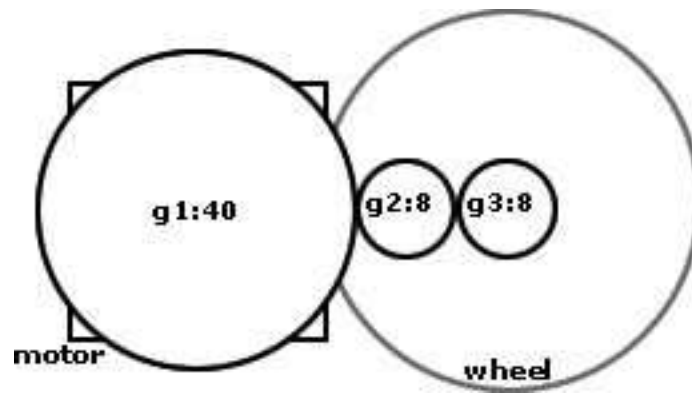
For each configurations below, determine how many times each gear rotates in a minute and in which direction (clockwise or counterclockwise). Assume that the motor is spinning clockwise at 200 rpm.



Answer:

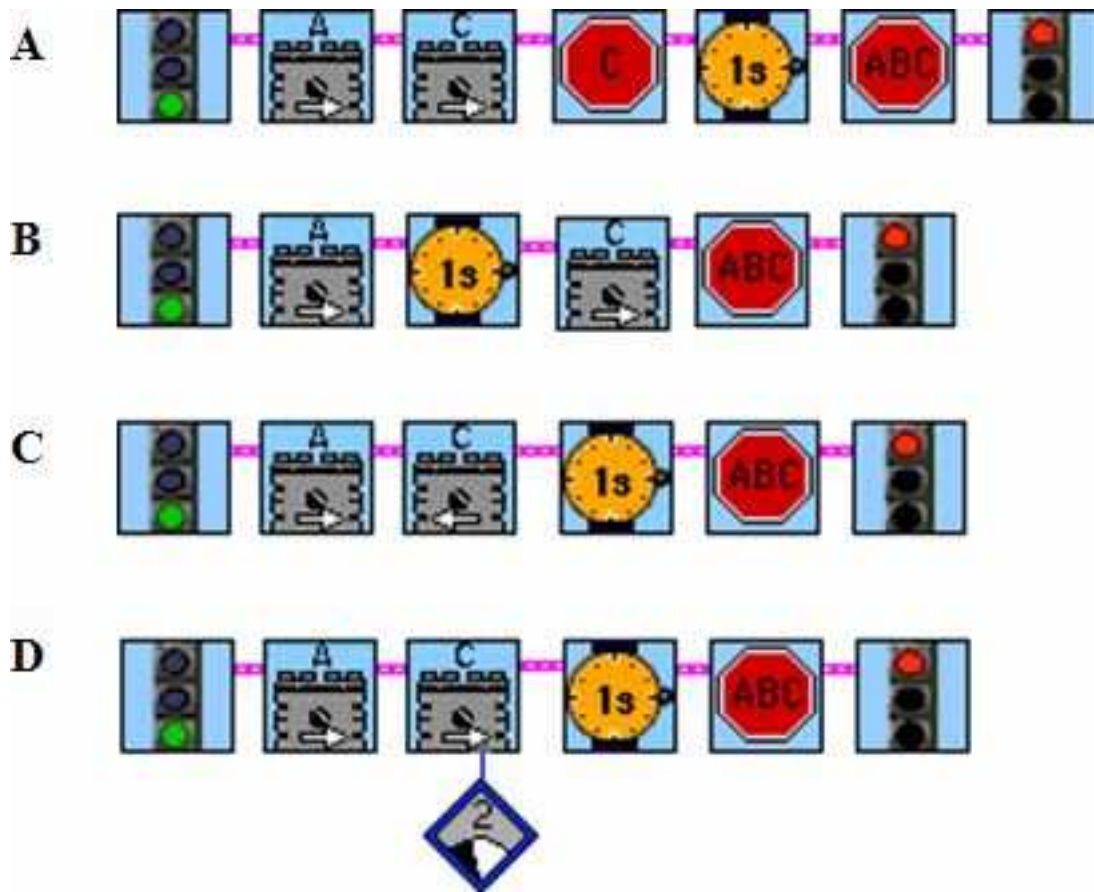


Answer:



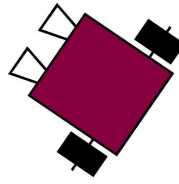
Answer:

program control



1. The programs shown above are similar in functionality. Explain briefly how each program runs and explain the differences between them.

2. Work with your group member(s) to implement each program.



Brooklyn The City
College University
of New York

Exploring Robotics (CORC 3303)

©2010

UNIT B Lab, part 2 : Robot Race

vocabulary

- gear ratio
- rotational speed
- robot control

materials

Make sure you have all of the following materials before you start the lab:

- lego robot kit
- communications tower

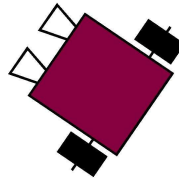
instructions

1. programming challenges

- Complete as many of the following programs as you can.
 - After you get each program to work, draw the code in the boxes provided and explain your approach (i.e.; physical design and robot control).
- (a) Program your robot to go as fast as possible for 4 second. Explain your physical design (i.e.; gear ratios and other modification)

- (b) Program your robot to complete the path, provided by the instructor, as fast as possible. Explain your physical design(i.e.; gear ratios and any other modification) and robot control (i.e.; smooth turn vs sharp turn).





UNIT C Lab, part 1 : Touch Sensors and Looping

vocabulary

- sensor
- touch sensor
- connecting lead
- modifier
- numeric constant
- loop
- multi-tasking

materials

Make sure you have all of the following materials before you start the lab:

- lego go-bot kit
- lego touch sensor and connecting lead

instructions

1. add a bumper and touch sensor to the go-bot

- Follow the **bumper sensor add-on instructions**, and work with your partner to add the bumper and touch sensor to your go-bot (as in the instructions).
- Add the **bumper sensor** to **port 1** of the RCX using the **connecting lead** :



connecting lead touch sensor

2. start up RoboLab

- Find the **RoboLab** icon on your computer and double-click on it to start it up.
Click on **PROGRAMMER**.
Double-click on **INVENTOR 4**.
Your screen will look like this:



- Now you are ready to start programming in RoboLab!

3. find the icons

- Locate the **functions palette**. See if you can find each of the following **icons**. What do you think each icon does? In the boxes below, next to each icon, write down your ideas.

4. loops

- Loops are structures that allow your program to repeat specific tasks again and again.
- All loops have the same basic structure. This includes a set of commands that you want the program to repeat and something that tells the program when to stop repeating. Some loops run for a specific number of times. These are the loops that we will discuss today. Other types of loops run until a certain **condition** occurs, for example, go forward until you hit a wall. We will discuss these types of loops another time.



- In RoboLab, loops begin with a **start of loop** icon and end with an **end of loop** icon



- There is also a **loop counter** which tells the loop how many times to repeat. This is specified

123

using the **numeric constant** icon.

- Note that if you do not assign a loop counter, the default is to loop (i.e., repeat the set of commands inside the loop) only twice.
- Inside the loop is the section of your program that you want to repeat.

5. programming challenges

- Complete as many of the following programs as you can.
- After you get each program to work, draw the code in the boxes provided on the next page.

(a) Consider the following example program:



- What do you think it does? Write down your ideas below:

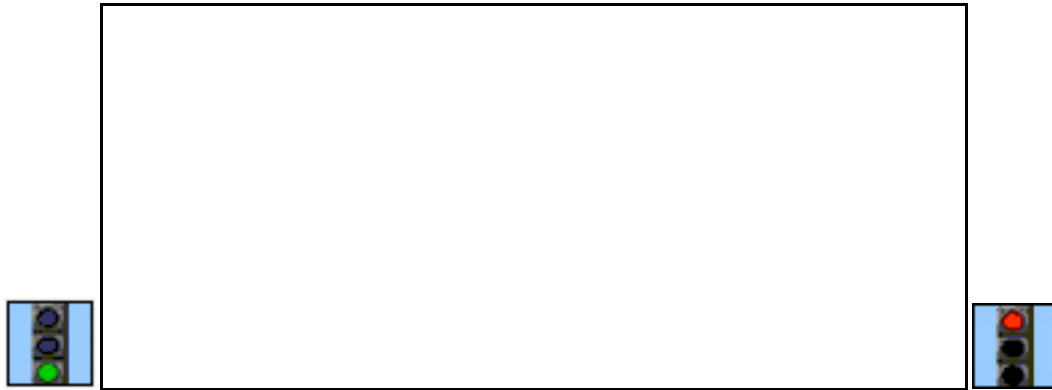
- Create the program in RoboLab. Test it and run it. What did it do? Did you predict right (above)?

(b) Modify the program using a loop structure:



- Does it behave the same as the program above or differently? Why or why not?

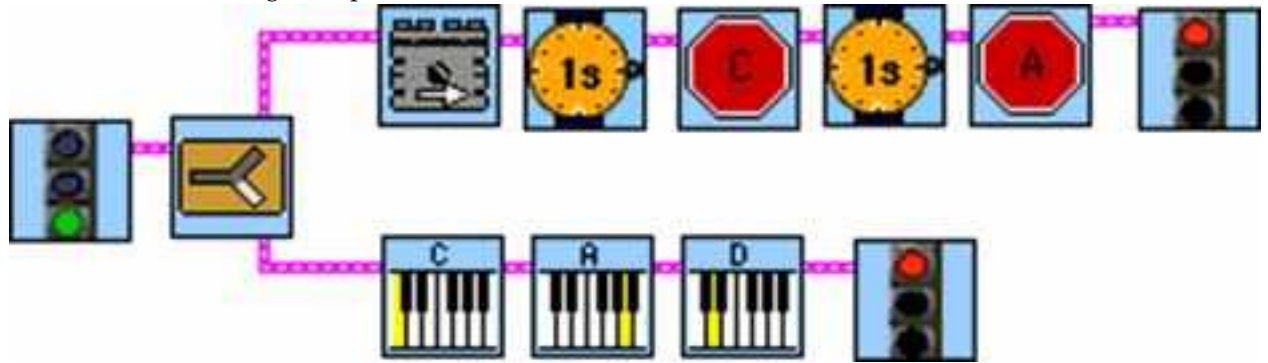
(c) Write a program to make the robot go in a square—but this time, use a loop structure.





6. programming challenges

- Complete as many of the following programs as you can.
- After you get each program to work, draw the code in the boxes provided on the next page.

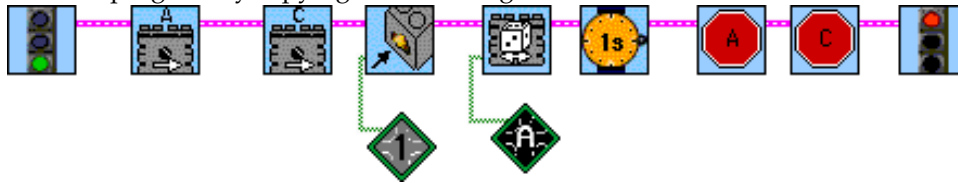
(a) Consider the following example:



- What do you think it does? Write down your ideas below:

- In RoboLab, you have been writing all your programs to have one **begin**  and one **end** . When you write a multi-tasking program, you still have one **begin**; however, at some point in the program, it will divide into multiple tasks, and each task will have its own **end**.
- Create the above example in RoboLab. Download and test it. Does it behave as described above?

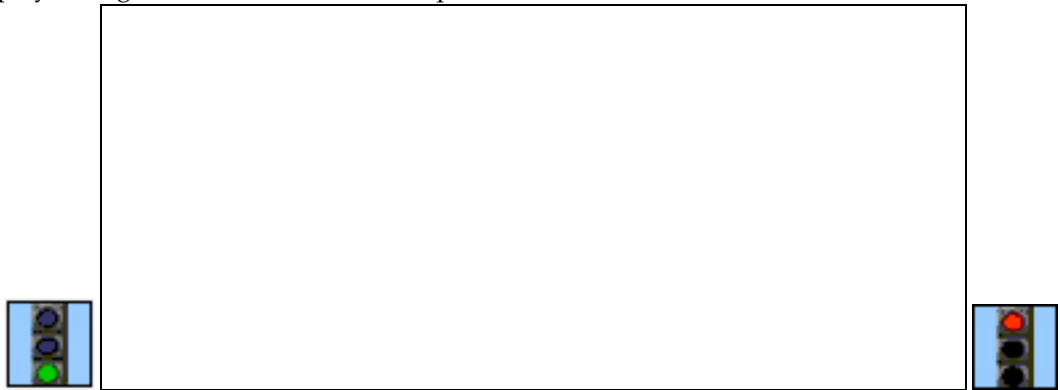
(b) Create a program by copying the following:



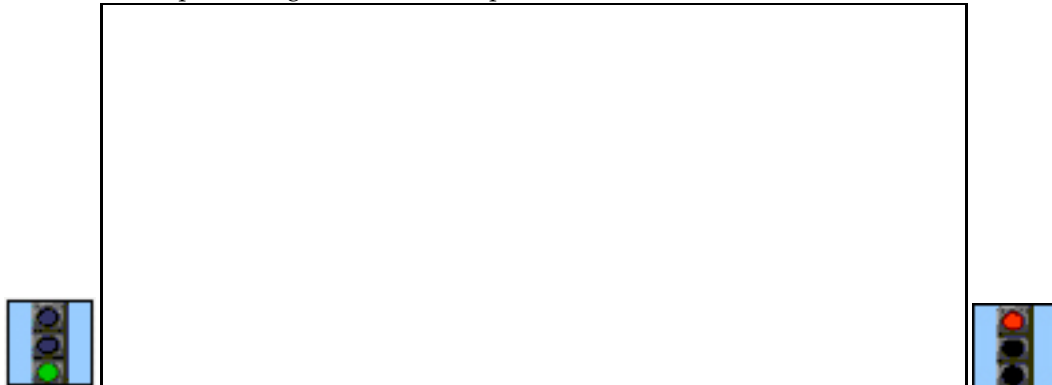
- What do you think it does? Write down your ideas below:

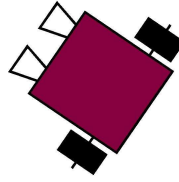
- **Download** your program and test it by running it several times.
- What did your robot do? Did it behave the same way each time you ran it? Write your answer below.

- Modify the program to keep going in a random direction until the touch sensor is pushed, play a song and then turn left and stop.



(c) Program the robot to go forward when the touch sensor is pushed, then turn in place until the touch sensor is pushed again and then stop.





UNIT C Lab, part 2 : RoboCupJunior Dance

1. Design a program for a robot that can DANCE to music.

You will be graded on the following criteria for your group assessment:

<i>criteria</i>	<i>points</i>
Robot Design and Construction (2 points)	
Programming and Preparation (2 points)	
Use of Sensors (1 point)	

2. You can also change the robot's body structure and dress it up in a costume!
3. During your lab session, plan out the robot's dance steps.

(a) First, pick a song for the robot to dance to. Name your song and artist.

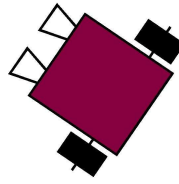
(b) Then, decide on the choreography. It may help to get up and walk through the moves yourself.

(c) Write down the moves you want your robot to perform. For example, go forward for 1 second, turn left and go forward again for 1 second.

- (d) Then translate your natural language (i.e., English) description of the steps (moves) into RoboLab commands. Draw the RoboLab icons here.

- (e) It is very important to plan your program ahead of time, otherwise you will waste time “hacking” and not really thinking about what you are doing. While it is okay to experiment to see what different steps will look like, be careful not to run out of time. Make sure that you and your group will be able to finish the challenge!
- (f) When your dance is choreographed on paper, you can use any remaining time to modify the robot’s body and make a costume for it.
- (g) After you finished your planning, create the program in RoboLab for the robot to dance to music.
- (h) What did your robot do? Is that what you expected? Write your answer below.

- (i) Go through the *iterative process* of *modifying* your program until it works! Then in the space below, draw your code:



Brooklyn College
The City University of New York

Exploring Robotics (CORC 3303)

©2011

UNIT D Lab, part 1 : Light Sensors and Forks

vocabulary

- light sensor
- fork
- merge
- jump
- loop

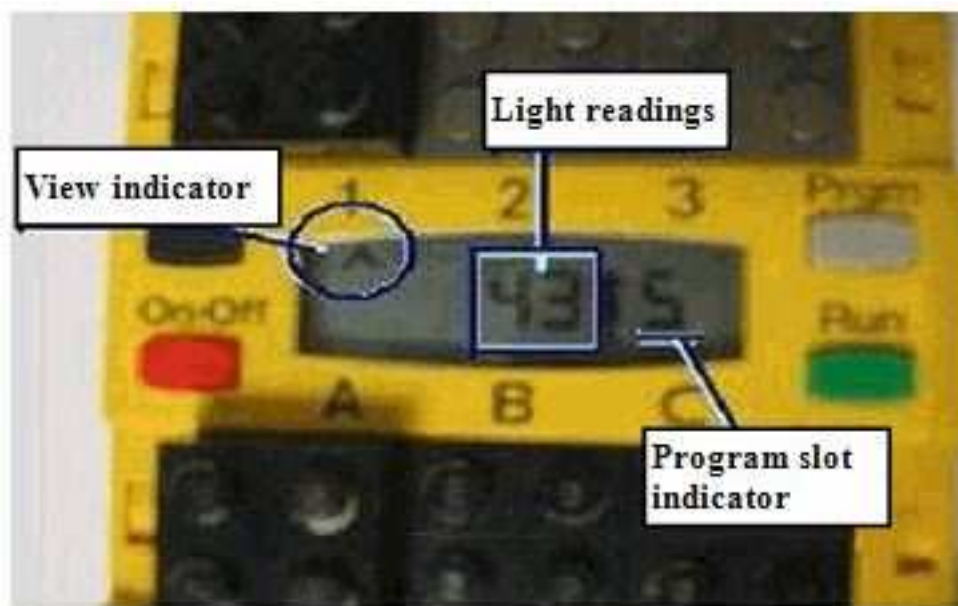
materials

Make sure you have all of the following materials before you start the lab:

- lego go-bot
- lego light sensor

light sensors

- The light sensor has a transmitter and a receiver. It transmits infrared light (also called "IR"), which bounces off objects and then returns in the direction of the receiver. The receiver records a value indicating how much light it read, which basically tells you about the brightness of the object that the light sensor is pointing at. The light sensor produces a value between 0 and 100, where 100 means very bright and 0 means very dark.
- Did you know that your TV remote control uses IR to talk to your television?
- **Here is how you can see what the light sensor's value is:**
This is done by turning on the RCX and pressing the **VIEW** button until the arrow at the top of the screen is under the port the light sensor is plugged into. For example, my light sensor is connected to port 2. I would turn on the RCX and press the VIEW button twice. Next to the image of the robot man, there is a number between 0 and 100—this is the value being read by the light sensor.



- Put your robot on at least four different locations: (a) white foamcore or paper, (b) black tape, (c) green tape, and (d) silver tape. View the different values for the light sensor and record the values in the table below:

	black	white	golden	silver
light sensor value				

decision-making

- Up to now, your programs have been sequential and linear. Basically, your programs have been executing each commands starting from the **green light** and going until the **red light**, without skipping any commands. For instance, we might like to program the robot to do one action (i.e., go forward) when some condition is true (i.e., light sensor sees black) or else do some other action (i.e., go backward) when the condition is not true (i.e., light sensor sees white). However, we need a **decision-making** mechanism so that our robots can react to their environment **autonomously** (i.e., without a human touching it). This decision making ability is based on its sensory inputs (e.g., the value of the light sensor) and will make the robot a little more intelligent! Decision making is achieved by **conditional execution** in the programming environment. In RoboLab, by using **fork** structures, we can allow programs to behave differently based on different values of sensor inputs.

- How does a fork work?

- When a fork is reached in a program, one of two **branches** will be taken based on the value read from one of the robot's sensors.
- All of the fork commands on the forks palette use the greater-than ($>$) or less-than-or-equal (\leq) condition to determine which branch to execute. These conditions are based on a **threshold**

 123

value. If the value read by the fork's sensor is greater than the threshold value, then the program follows the top branch. Otherwise, if the value read by the fork's sensor is less than or equal to the threshold value, then the program follows the bottom branch. For example, RoboLab can let a program make a decision depending on the value it gets from the **light sensor**



when you use the **light sensor fork**

- You must attach the correct port number modifier (i.e., port 1  , port 2  or port 3

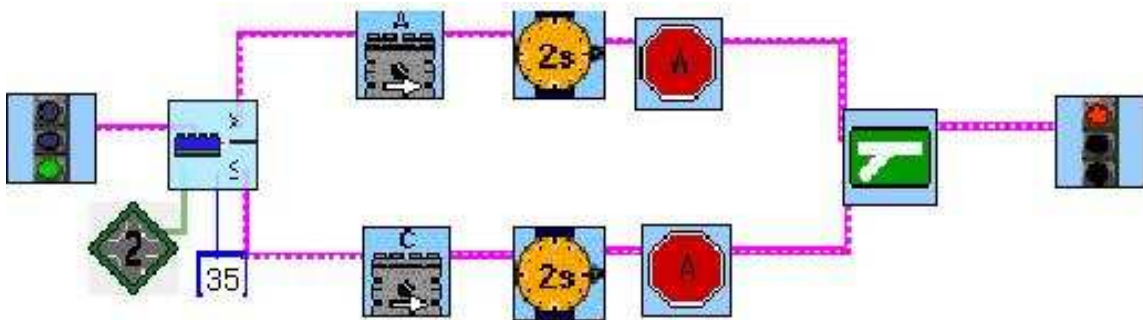
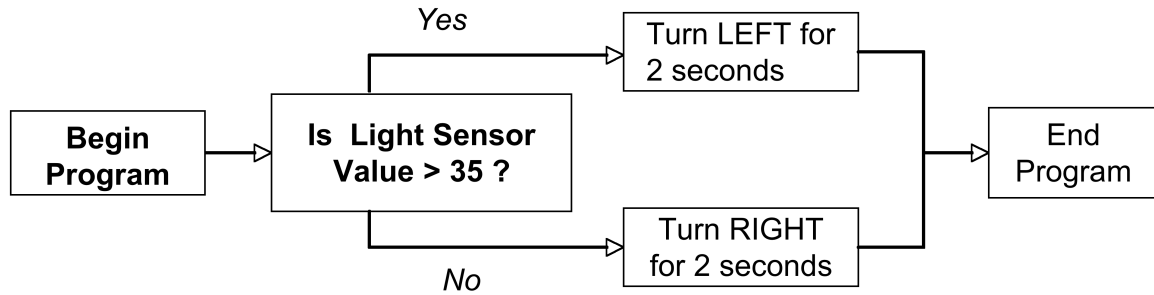


) to the fork where your actual sensor is connected.



- A **fork merge** is required at some point in the program with any fork command. It completes your conditional structure and brings the two branches of the program back together.

- Consider the following program:



- Create this program in RoboLab and use the light sensor value returned when the robot sees black tape as the threshold value. **REMEMBER:**
 - * The light sensor fork requires you to specify a **threshold** value. For instance, the above example uses a light sensor fork with a threshold value of 35—but you need to **calibrate** your robot to determine which value it reads when it sees black tape, and then use that value instead of 35.
 - * Don't forget to attach the correct port number to the fork. For example, our light sensor is attached to **port 2**.





putting it all together

1. Add a light sensor to the go bot.

- Attach a light sensor to the front of your go-bot. Make sure it is looking down and attached as closely as possible to the ground.
- Connect the light sensor to port 2 of the RCX with a connecting lead.


2. Find the icons.

- Locate the functions palette. See if you can find each of the following icons. What do you think each icon does? Write down your ideas in the boxes below, next to each icon.


			
			

3. Jumps



- Unlike the loop , which repeats a certain number of times based on a counter or a condition, a jump repeats forever. This is also called an *infinite loop*.



- In RoboLab, a jump is created by using a land  icon in place of a start-of-loop icon and a



jump  icon in place of an end-loop icon.

- The jump icons comes in several colors: red, blue, yellow, green and black. Make sure to use the pair that match in color.
- Jump icons will run forever. If you use them wrong, the only way to stop the program is to turn off the RCX.





- Remember to place the land  icon BEFORE the jump  in your programs!

4. Light Sensor Fork

- A LEGO light sensor, when attached to the RCX, emits *InfraRed (IR)* signal and reads values indicating the amount of light that bounces back from the surface that it is directed at (this is called “active mode”).
- The light sensor returns a value between 0 (dark) and 100 (bright).

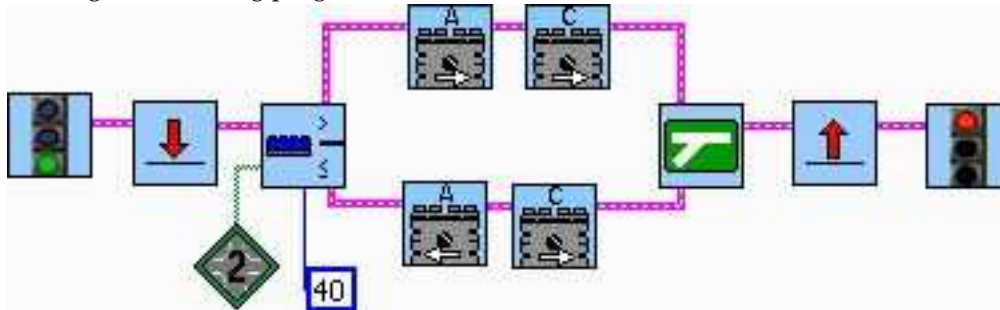


- In order to perform different actions based on light sensor values, the light sensor fork  icon can be helpful. This icon compares the values read by the light sensor to a user-specified (or default) value and chooses a branch of the program to execute.
- The light sensor fork has 2 modifiers:
 - (a) Port: corresponds to the RCX port number to which the sensor is connected.
 - (b) Compare to: a numerical constant modifier that holds the value to compare with the light sensor reading. If this modifier is not specified, the default value, 55, is used.

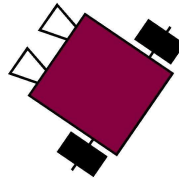
- Each fork branch needs to be connected with a fork merge  icon before the program terminates.

5. Create your program

- Use the mouse to select, drag and drop RoboLab icons onto your canvas and wire them together, creating the following program:



- What do you think it does? Write down your ideas below:



Brooklyn The City
College University
of New York

Exploring Robotics (CORC 3303)

©2010

UNIT D Lab, part 2 : Obstacle Avoidance

vocabulary

- touch sensor
- bumper
- fork
- branch
- jump
- lamp

materials

Make sure you have all of the following materials before you start the lab:

- lego go-bot
- lego touch sensor

instructions

1. add a bumper and touch sensor to the go-bot

- Follow the **bumper sensor add-on instructions**, and work with your partner to add the bumper and touch sensor to your go-bot (as in the instructions).
- Add the **bumper sensor** to **port 1** of the RCX using the **connecting lead** :



touch sensor

2. add a lamp to the go-bot

- Add a lamp to your robot. Connect it to output **port B**. You can plug it in directly or you can use a connecting lead.



lamp

3. start up RoboLab







- Find the **RoboLab** icon on your computer and double-click on it to start it up.
Click on **PROGRAMMER**.
Double-click on **INVENTOR 4**.
Your screen will look like this:




- Now you are ready to start programing in **RoboLab**!

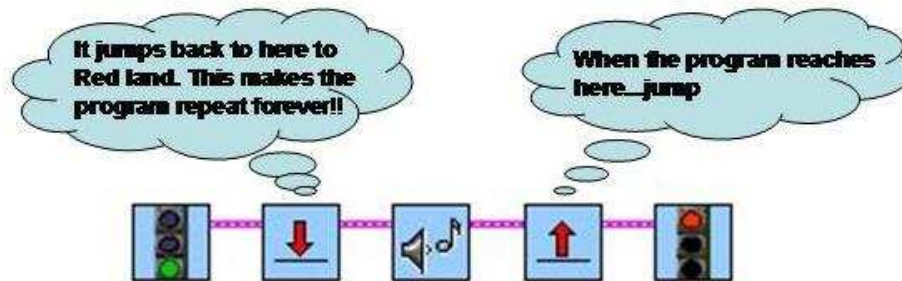
4. find the icons



- Locate the **functions palette**. See if you can find each of the following **icons**. What do you think each icon does? In the boxes below, next to each icon, write down your ideas.



					
					

5. jumps



- Unlike the **loop** , which repeats for a certain number of times based on a **counter** or a **condition**, a **jump** repeats forever. This is also called an **infinite loop**.



- In RoboLab, a **jump** is created by using a **land**  icon in place of a **start-of-loop** icon and a **jump**  icon in place of an **end-of-loop** icon.
- The **jump** and **land** icons come in several colors: red, blue, yellow, green and black. Always make sure that your colors match up! In other words, if you use a yellow land, you must also use a yellow jump to go with it.
- Note that with a **jump**, you do not have to include a modifier telling it how many times to repeat.
- However, it is recommended to use jumps carefully since your program will never stop! (until you turn off the RCX).

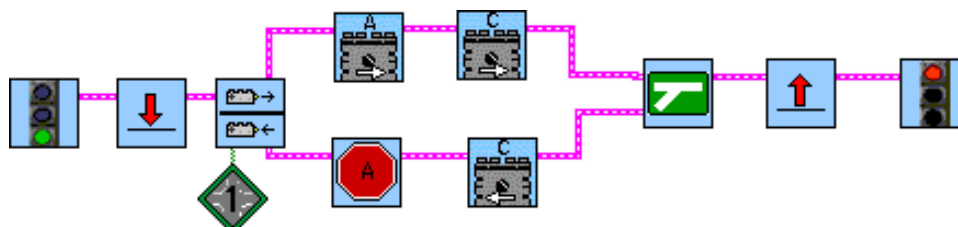
- Make sure you put the **land**  icon FIRST, BEFORE the the **jump**  icon. Otherwise, strange things may happen!

6. touch sensor fork

- A **touch sensor fork**  works similarly to a **light sensor fork** —it helps the robot make decisions about how to behave based on input from one of its sensors.

7. create your program

- Use the mouse to select, drag and drop RoboLab **icons** onto your canvas and **wire** them together, creating the following program:

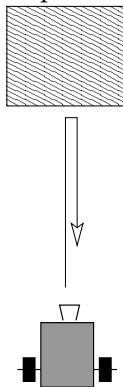


- What do you think it does? Write down your ideas below:

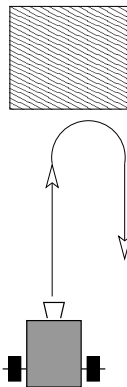
- Download and test your program.
 - If you have trouble, check these things:
 1. The RCX must be turned on during the download.
 2. The black part of the RCX should be turned towards your tower.
 3. If there is a lot of light in the room, try covering your RCX and the tower with a piece of paper.
 4. If your download arrow is broken, then you have an error in your program. Make sure that you copied all the icons correctly and that the wires all start and end in the right places.
 5. Make sure that the **bumper sensor** is connected to **port 1**.
- What did your robot do? Is that what you expected? Write your answer below.

8. obstacle avoidance

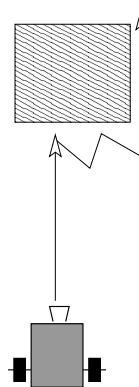
- One of the classic tasks that mobile robots perform is known as **obstacle avoidance**. Basically, this is a behavior in which robots avoid bumping into things they don't want to bump in to, in other words, obstacles.
- The way that robots do this is actually by bumping into obstacles to discover that they are there, and then backing up and/or turning around to avoid them.
- Some examples are shown below:



(a) robot backs up when it hits the obstacle



(b) robot turns around when it hits the obstacle

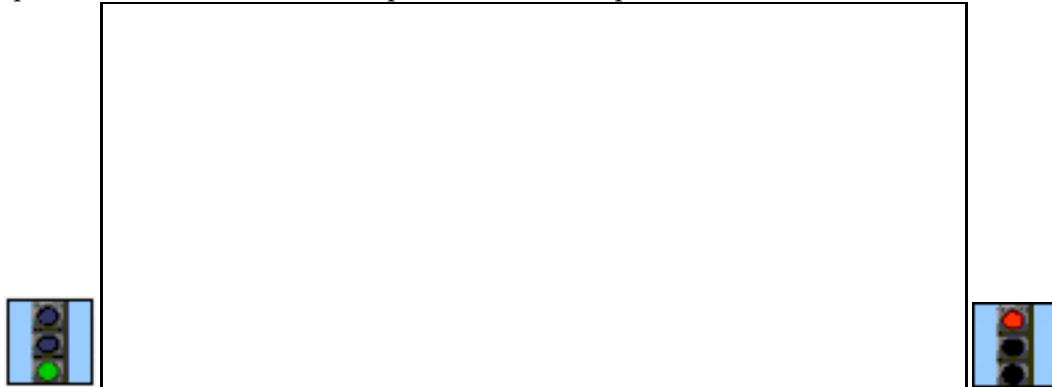


(c) robot backs up and edges its way around the obstacle

9. programming challenges

- Complete as many of the following programs as you can.
- After you get each program to work, draw the code in the boxes provided on the next page.

- (a) Program the robot to go forward until it bumps into something, and then turn on the lamp, back up for three seconds and then stop. This is like example (a) above.

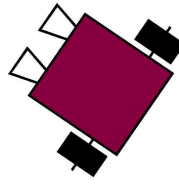


- (b) Program the robot to go forward until it bumps into something, and then turn on the lamp, turn around and go back the way it came, stopping after three seconds. This is like example (b) above.



- (c) Program the robot to go forward until it bumps into something, and then back up for one second and go forward again until it bumps into something, then back up and go forward again—repeating this behavior forever. This is like example (c) above.





Brooklyn College
The City University of New York

Exploring Robotics (CORC 3303)

©2010

UNIT E Lab : Multi-tasking

vocabulary

- task
- multi-tasking
- hardware conflicts
- obstacle avoidance
- line following
- bumper
- infra-red light (IR)
- calibration
- fork
- branch
- threshold value

materials

Make sure you have all of the following materials before you start the lab:

- touch sensor
- two light sensors
- lamp

instructions

1. add a bumper and touch sensor to the go-bot

- Follow the **bumper sensor add-on instructions (page 13, lego construction booklet)**, and work with your partner to add the bumper and touch sensor to your go-bot (as in the instructions).
- Add the **bumper sensor** to **port 2** of the RCX using the **connecting lead**

2. add two light sensors to the go-bot

- Work with your partner to add two light sensors to your go-bot (as in the instructions).
- Add the first **light sensor** to **port 1** of the RCX using the **connecting lead**
- Add the second **light sensor** to **port 3** of the RCX using the **connecting lead**







3. add a lamp to the go-bot

- Add a lamp to your robot. Connect it to output **port B**. You can plug it in directly or you can use a connecting lead.

4. start up RoboLab

5. find the icons

- Locate the **functions palette**. See if you can find each of the following **icons**. What do you think each icon does? In the boxes below, next to each icon, write down your ideas.



					
					


6. multi-tasking

- Sometimes you may want to have your robot do two things at once. This is called **multi-tasking**. For example, if you were programming your robot to play the game of soccer, then you would want the robot to be able to look for the soccer ball and at the same time, you would want it to look out for obstacles.
- We human beings are superb at **multi-tasking**! You can *listen* to your favorite music while *walking* down the street and *eating* candy all at the same time. In fact, we are always multi-tasking—hearing and seeing and feeling and breathing all at once.
- Your computer is also **multi-tasking** by having multiple windows open at once. You can browse in the internet (e.g., using Mozilla) and play music (e.g., running iTunes) at the same time.
- The RCX also has the ability to multi-task, meaning it can execute more than one task at a time.

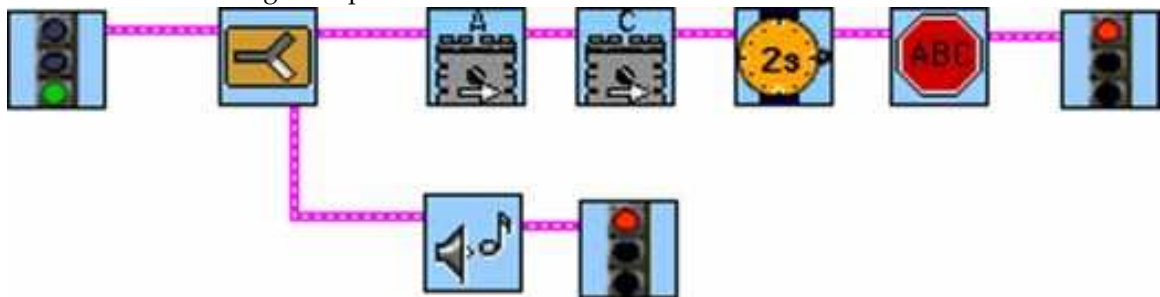


In RoboLab, the **task split** icon is used to achieve this. This structure allows you to run two different tasks at the same time. This is very helpful for monitoring two sensors at the same time. For example, you can assign one task to handle the touch sensor while another task handles the light sensor. The RCX can actually run up to 8 tasks simultaneously!

- In RoboLab, you have been writing all your programs to have one **begin**  and one **end** . When you write a multi-tasking program, you still have one **begin**; however, at some point in the program, it will divide into multiple tasks, and each task will have its own **end**.
- Note that this is different from a **fork**, because you'll remember that forks divide a program into

two branches, and the branches come back together using a **fork merge** .

- With multi-tasking, the branches (which are called **tasks** in this context) do not come back together.
- Consider the following example:



- What do you think it does? Write down your ideas below:

- Create the above example in RoboLab. Download and test it. Is that what you expected??

7. hardware conflicts

- The ability to run more than one program at a time also has a problem since tasks can share the same hardware resources. If one task asks Motor C to go backward and the other asks it go forward at the same, we have a **hardware conflict**.
- You need to think about possible hardware conflicts when you write a multi-tasking program. Make sure that multiple tasks do not try to control the same hardware at the same time.

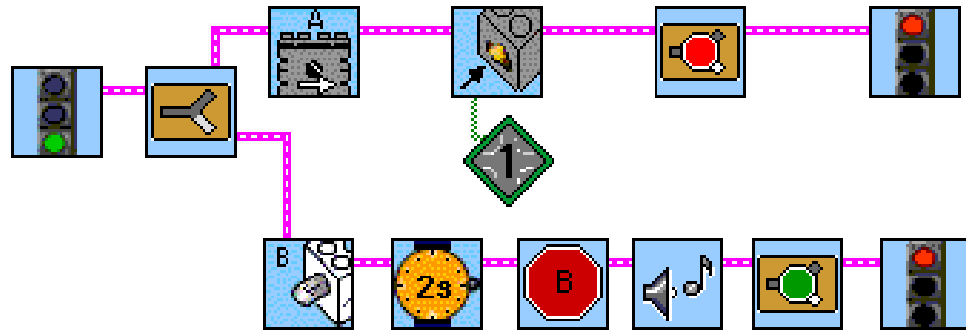
- You can use the **start task**  and **stop task**  icons to handle possible hardware conflicts.



- The **start task** icon causes the program to restart tasks after they have been completed or stopped. If you want to restart a certain task and not all tasks, then you can specify a **task number** using a modifier.



- The **stop task** icon causes the program to stop tasks that may be still in progress. If you want to stop a particular task and not all tasks, then you can specify a **task number** using a modifier.
- Consider the following example:



– What do you think it does? Write down your ideas below:

- Create the above example in RoboLab. Download and test it. Does it behave as described above?

8. things to remember about tasks

- *You can not merge tasks.* You can only merge forks.
- A task split is basically the same thing as having another program running.
- If you use a task split, each separate program **MUST** have its own stoplight.
- Be careful of hardware conflicts when two tasks try to control a single output at the same time.
- You can not use a **jump** to jump between tasks.
- **Never** use a **jump** to jump from one side of a task to the other side. Basically, don't jump over task splits.

9. line following

- Another classic task for robots is called **line following**. Here, the robot is placed on a white floor and given a black line to follow.

- The way that robots do this is by actually following along the edge of the line, on either side where black meets white, rather than following the line itself.
- The robot has a light sensor pointed toward the floor, and it goes along reading the value of the light sensor. It will assume that the value should be black (say 35), so if the value changes to something higher (say 45), then the robot should turn a little bit until it reads black again. An example is shown below:



10. programming challenges

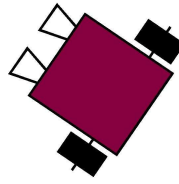
- Complete as many of the following programs as you can.
- After you get each program to work, draw the code in the boxes provided on the next page.

(a) Program the robot to follow a black line.



(b) Program the robot to go forward if it is on a black line and to light the lamp and play a short song when it bumps into something.





Brooklyn College
The City
University
of New York

Exploring Robotics (CORC 3303)

©2010

UNIT F Lab : RoboCupJunior Soccer

vocabulary

- RoboCupJunior soccer field
- RoboCupJunior electronic soccer ball

materials

Make sure you have all of the following materials before you start the lab:

- lego go-bot with bumper sensor and light sensor
- RoboCupJunior soccer field
- RoboCupJunior electronic soccer ball

instructions

1. modify the robot's the light sensor

- Modify the robot's light sensor so that it is point straight out in front of the robot, instead of pointing toward the floor. It will use this to see the electronic soccer ball, which transmits infra-red (IR) light. The light sensor will read this as a very bright value, probably returning a value greater than 55.
- Keep the second light sensor of the robot looking to read the floor and tell whether it is heading toward the "white" goal or the "black" goal.

2. start up RoboLab

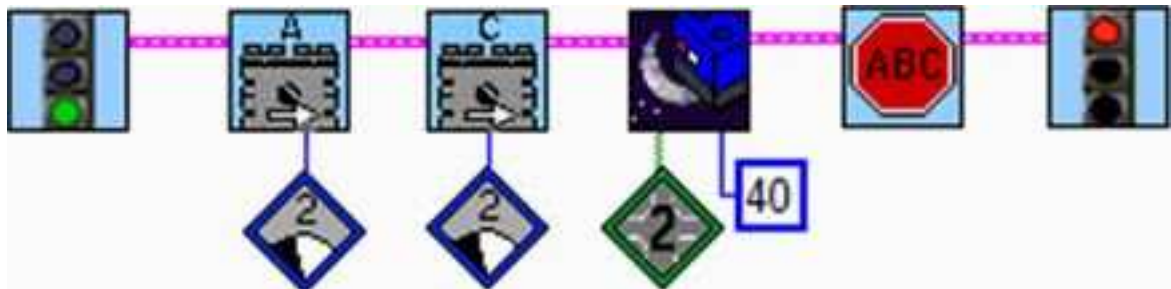
3. find the icons

- Locate the **functions palette**. See if you can find each of the following **icons**. What do you think each icon does? In the boxes below, next to each icon, write down your ideas.

4. create your program

- Use the mouse to select, drag and drop RoboLab **icons** onto your canvas and **wire** them together, creating the following program:



5. download your program

6. run the program

- What did your robot do? Is that what you expected? Write your answer below.

7. programming challenges

- Program the robot to play a simple game of soccer using two tasks. One task should make the robot spin around until it sees the soccer ball, using a light sensor pointing forward and looking for something very bright; then the robot should move toward the bright object. The second task should make the robot turn around and move away if it bumps into anything.

