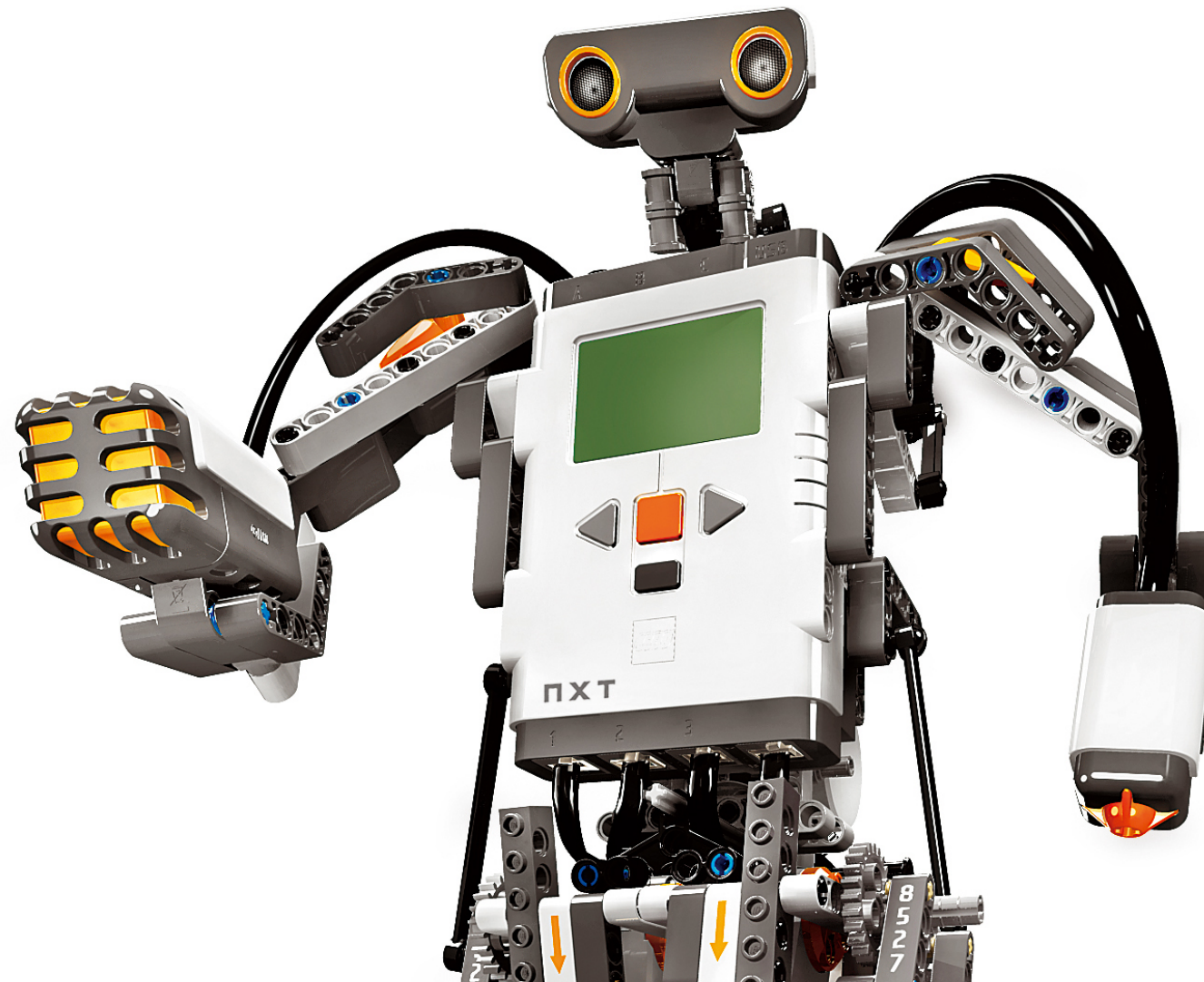


Robotics with Enchanting and LEGO® NXT

A Project Based Introduction to Programming

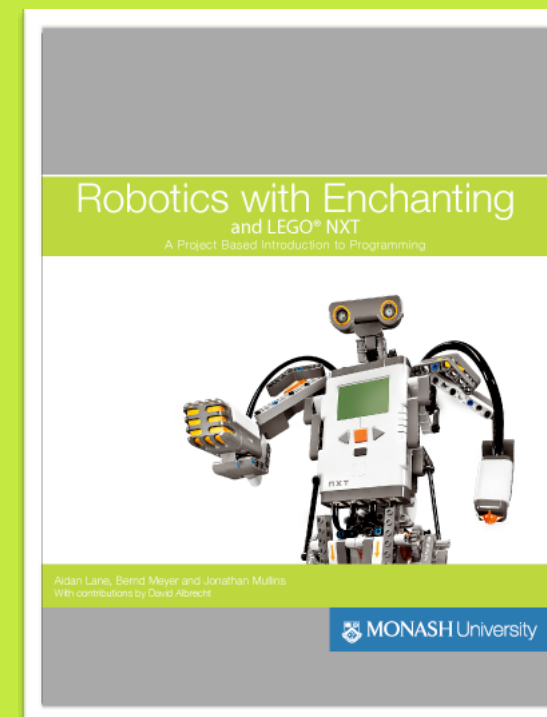
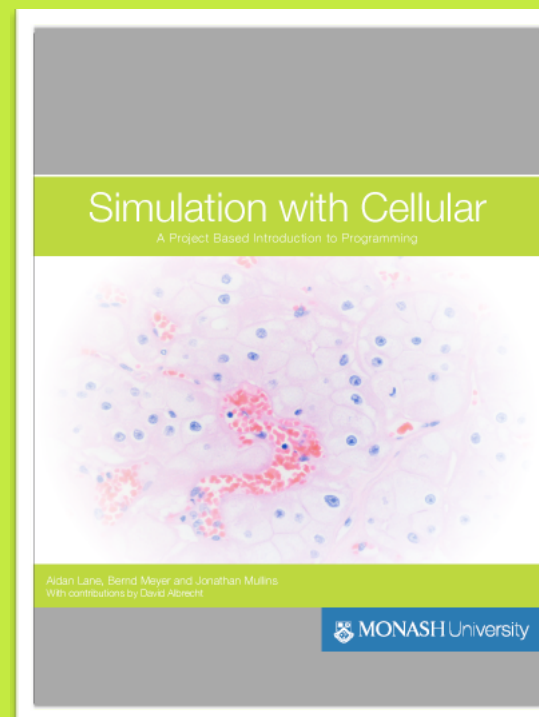
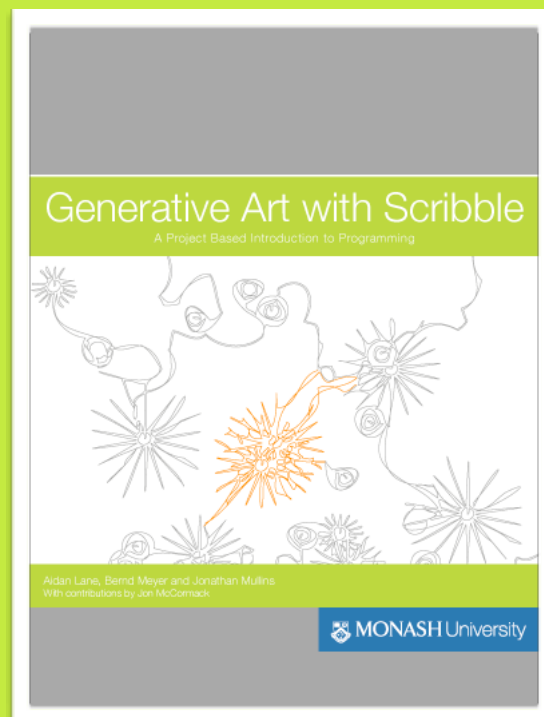


Aidan Lane, Bernd Meyer and Jonathan Mullins
With contributions by David Albrecht



MONASH University

BlockBooks Series



Simulation with Cellular

Cover photo is an adaptation of “[Renal Cell Carcinoma, Chromophobe Type](#)”

© 2012 [cnicholsonpath](#), [CC BY 2.0](#)

Robotics with Enchanting and LEGO® NXT

Cover photo © The LEGO® Group. All Rights Reserved.

Like to be notified about updates and new releases?

Register Here!

Copyright

Robotics with Enchanting and LEGO® NXT
A Project Based Introduction to Programming

Aidan Lane, Bernd Meyer and Jonathan Mullins
With contributions by David Albrecht

Version 1.1

Except where otherwise noted, this book, including all adaptations of photos and imagery, is © 2012 [Monash University](http://monash.edu) under a Creative Commons Attribution-ShareAlike license:

<http://creativecommons.org/licenses/by-sa/3.0/>

LEGO® and MINDSTORMS are trademarks of the LEGO Group, which does not sponsor, authorise or endorse this publication.

All other trademarks are the property of their respective owners.
All rights reserved.

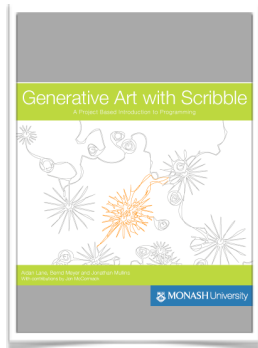
Cover photo © The LEGO® Group. All Rights Reserved.

Foreword

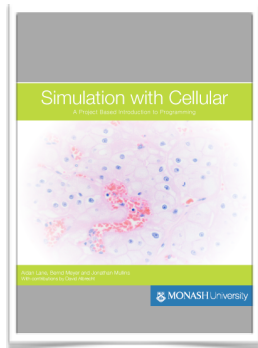
Welcome to the BlockBooks Series!

The three books in this series provide you with a project-based introduction to programming which you will hopefully enjoy as much as we enjoy developing it.

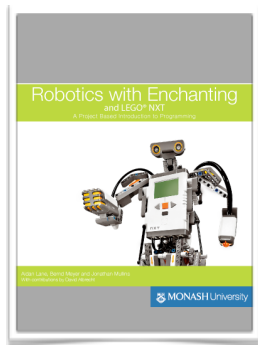
The series was born from the firm conviction that your first steps in programming can be easy and exciting rather than dull and difficult. We use a project-based approach which takes you on a guided tour of discovery of the principles of programming. Such experiential learning makes the fundamental ideas very accessible. The BlockBooks series currently consists of three different volumes which can be used individually but also complement each other well. Each book by itself will provide you with a first introduction to programming. Using all three you will benefit even more as they highlight different aspects.



Generative Art with Scribble approaches programming through generative arts. Instead of viewing an artwork just as a final (static) product we focus on the process that creates it, then automate this. Can a programmatically produced picture be art? You be the judge... and the artist!



Simulation with Cellular introduces programming through the idea of simulations. You get to construct and explore computer simulations of a variety of natural and social phenomena, such as the behaviour of bacteria, the spread of wildfires, and urban settlement. Along the way you will also find out what all these things have in common!



Robotics with Enchanting and LEGO® NXT uses robotics as the path to programming. How do you teach a robot to cope with the challenges it encounters? You will discover the solution to fundamental problems, such as how a robot can follow a visual clue, and get to work on tricky problems, such as designing a champion program for a Sumo robot wrestling competition. To perform the activities in this book you need access to a LEGO® NXT robot.

Pick any book that appeals to you to start your journey into programming.

The approach and philosophy of these materials deviates significantly from the traditional introduction to programming. Most programming books will run you through the technicalities of a specific programming language in bite-sized pieces so that you have to read hundreds of pages before you can even think of tackling any reasonably sized project. Our project-based approach instead aims to let you "get your hands dirty" right from the start. We provide you with a series of increasingly complex projects. Each project is supported with a number of short video tutorials that give you just enough knowledge to tackle the project without giving away the solution. In this way you won't have to wade through hundreds of pages before you get to experience programming. You learn by doing, and we will give you just enough support to enable you to discover the fundamental principles yourself. This is not only much more fun but also leads to a far deeper understanding. It does, however, require your motivation to embark on and persist with this journey of discovery. If you passively consume the tutorials without undertaking the activities, your learning will be very limited.

The second crucial aspect that differentiates these materials from traditional programming introductions is the choice of programming language. An industrial strength programming language, such as Java, C, or Python, puts a significant number

of hurdles into your way. There are a very strict syntax to memorise, a large library of functions to understand even to solve simple tasks, and often complex programming environments to use. While all of this complexity is necessary for the production of large software systems and important to master for a professional programmer, none of it helps you to understand the fundamental principles of programming. Indeed, it stands in the way of seeing these.

Studying the basic concepts rather than the technicalities of programming provides you with some real benefits even if your goal is not at all to program a computer. Programming simply means to devise an exact methods to solve a problem and to describe this method with sufficient precision for the rigid "mind" of a computer to understand. This is the essence of "computational thinking." The structured way of thinking about problems that you acquire in doing so is an enormously powerful tool that will help you to solve problems you may have deemed too complex before. Being able to describe your methods precisely will also make you an effective communicator of such methods.

To avoid overloading you with technical minutiae of industrial programming languages we have chosen to base our materials on an educational language called [BYOB](#), which was developed at the University of California in Berkley. This language was designed specifically for educational use and removes a

significant number of hurdles by providing you with a very clearly structured visual language and a simple programming environment that supports you in composing correct programs. The basic idea is to (graphically) construct programs from simple building blocks by snapping together blocks that encapsulate basic functionality. Imagine building with LEGO® blocks. I won't say more as you will quickly discover what this means once you start working with the books. But make no mistake: while all of this may sound like child's play, visual block-based programming languages are now rapidly becoming popular beyond educational use. For example, Google introduced a programming language called [AppInventor](#) to program Android smart phones (now maintained by MIT), and the commercial environment [Stencyl](#) uses the same principles for iOS game development.

It is hard to convey the sense of excitement that our team felt when we discovered BYOB. We had previously been developing a secondary curriculum for introductory computer science that was first road-tested at the [John Monash Science School](#) in the [Emerging Technologies Curriculum](#). Convinced that an industrial main stream language is not the right vehicle for teaching first steps in programming we had been experimenting with a number of special purpose educational languages, including [StarLogo TNG](#) which uses a similar visual block-based approach. Our experiences with these languages were very positive, but it was impossible to get around the fact that these special purpose languages are very limited. Students quickly "hit the wall" if they

want to implement their own ideas beyond what the language was built for. This can lead to significant frustration, specifically in highly motivated learners, and leads to difficulties with continuity in the curriculum.

Enter BYOB. The beauty and ingenuity of BYOB is that its developers maintained the simplicity of the visual block-based languages but removed the limitations. BYOB is a fully fledged, open-ended language that allows you to do anything that any other programming language would allow you to do. It does not set you limits. However, the entry threshold still remains very low. When our team first discovered BYOB we immediately knew that we had found exactly the tool that we were missing. However, there was one limitation: pure BYOB does not offer direct support for the types of projects that we wanted students to undertake – the control of robots, the implementation of simulations and the creation of visual artworks. At Monash, we thus set about to construct purpose-built extensions of BYOB that support these goals directly. The result is a suite of programming languages that will allow you to tackle such fun projects with the same ease as special purpose languages but that will never set you a limit in what you can program or how you can do it. These are the languages "Cellular", "Enchanting", and "Scribble" used as the basis of the three books.

At Monash we will start to use BYOB as the entry point to teaching programming from 2013 and a number of Australian

schools that participate in the world-wide [Google CS4HS](#) (Computer Science for High Schools) initiative are adopting these systems for their own IT curricula.

A word of warning is in place. At the current point of development, the books are not fully self-contained comprehensive introductions to programming. The books were designed primarily as ["flipped classroom"](#) support materials where learning is facilitated by a teacher. In this scenario, the teacher will select (and maybe extend) the projects, students will watch the tutorials and will then (ideally in groups) work through the projects supported by their teacher. Classroom group discussions are subsequently used to reflect on the different approaches to the task. However, we believe that the materials are also suitable for completely self-guided study by motivated students. In this case, it is advisable to work through the activities and challenge projects in the given order and to watch the associated video tutorials before tackling each of them. The difficulty of the projects is quite finely graded so that it is unlikely that you will get stuck completely. Having to crack the occasional harder nut along the way is part of the fun. However, if you do get stuck, there are online discussion forums tied into the book where like-minded students get together to collaboratively solve the challenges, to help each other, and to discuss their approaches.

A significant number of people have, in direct and indirect form, influenced the development of the materials you have in front of you.

First, thanks have to go out to Jens Mönig and Brian Harvey, the brains behind BYOB. Without them and the volunteers who have contributed to the BYOB developments, none of this could have happened. You are a true inspiration!

The inimitable Jonathan Mullins has single-handedly designed and produced all of the video tutorials and many other materials, not to mention the fact that he implemented Cellular, Scribble, and much of the port of Enchanting to BYOB. Aidan Lane has expertly implemented the iBooks, constantly pushing the boundaries of interactive functions in eBooks with his habitual ingenuity. David Albrecht and Jon McCormack have helped to shape the materials in countless discussions. So did the Emerging Technologies and Creative Curriculum team at JMSS, specifically Neil Carmona-Vickery, Victor Rajewski, and Linda McIver, who shared our enthusiasm and had the courage to road-test these ideas. The feedback gained in this way was invaluable.

Cagatay Goncu contributed in the most fundamental way: It is impossible to forget how he conjured up the first prototype of BYOB for robotics. I floated the idea of controlling robots with BYOB with him late one evening and we speculated together how this could best be achieved. When I arrived back at the office the

next morning Cagatay had stopped to speculate. Instead, he had created the first functional prototype literally overnight! Without this initial impetus we might never have taken the idea further. Together with Nizam Abdallah he continued to develop this first prototype. We later discovered that the Southern Alberta Robotics Enthusiasts were pursuing a related goal, namely to create a version of [Scratch](#) to control [LEGO® NXT](#) robots. As Scratch is the basis of BYOB it seemed natural to join forces. This led to Clinton Blackmore and Jonathan Mullins collaborating on porting Enchanting from Scratch to BYOB.

Finally, thanks to the leadership of the Faculty of Information Technology at Monash University and at John Monash Science School for sharing the vision and for supporting the development of an experimental curriculum: Ron Weber, Peter Corkill, and Andrew Chisholm.

Numerous others have contributed in various form, and I apologise to everyone who has not been mentioned.

All our developments are open source software and shared under creative commons. We will continue to develop these materials and we hope that a growing community will join us in this effort.

[Please let us know](#) if you decide to use these materials in the classroom.

Bernd Meyer

Melbourne, October 2012

Chapter 1

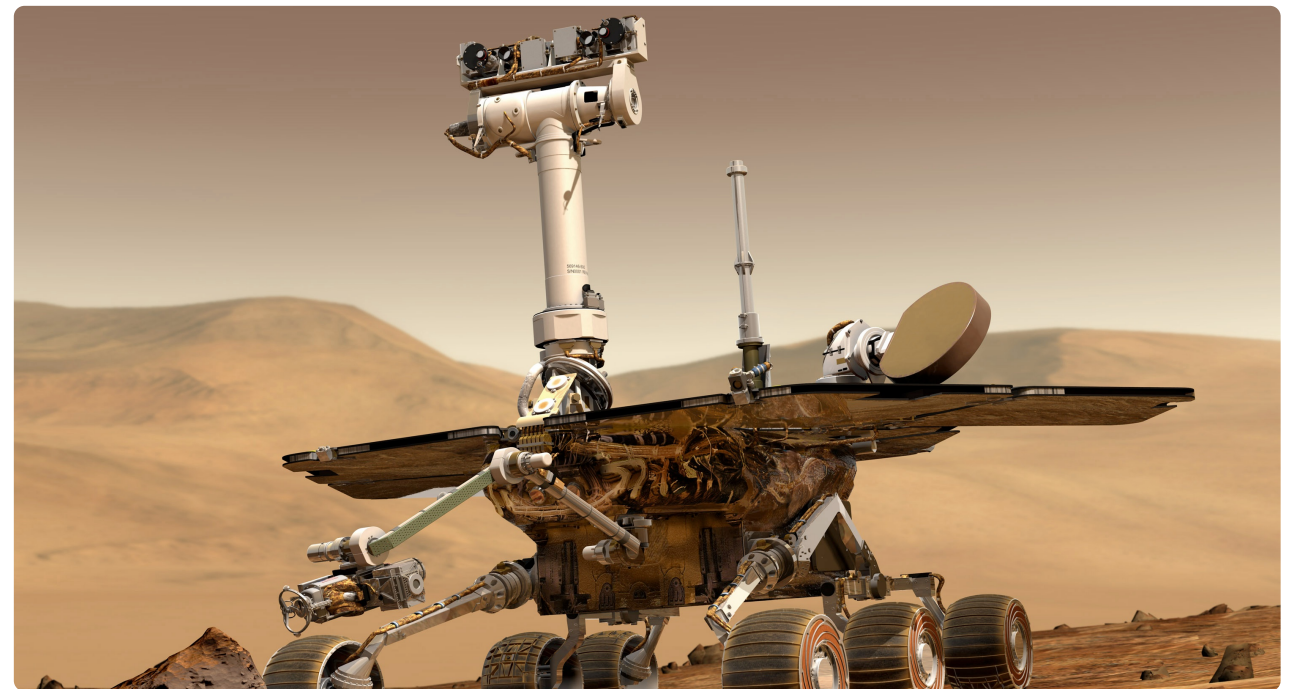
Introduction

Section 1

Robotics

The use of robots to perform meaningful work is becoming increasingly prevalent throughout the world. From automated vacuum cleaners to mobile science labs wandering around the surface of Mars, it's clear that there are very few human endeavours that can't benefit from the help of a robot.

Figure 1.1 Robots in the Wild



NASA's Mars Exploration Rover
Courtesy of NASA/JPL-Caltech

While the present state of technology does not allow for robots to think with the creativity and intelligence of the human brain, there are many tasks which are better suited to the “brain” of a computer, which is reliable, predictable, and devoid of emotion and temperament.

Jobs involving manual labour are already being replaced by robots, for example motor vehicle production lines and food packaging plants. It's likely that in the future we'll see more and more robots in our environment, perhaps packing the shelves at the supermarket, repairing potholes in the road, cleaning public spaces, and practically any industry where a work task can be clearly defined by a set of instructions.

What is a robot?

You might assume that a robot has to be a machine that walks and talks, maybe with two blinking lights for eyes and a monotone voice, but in fact the definition of a robot is far more general than that. There is no absolute consensus on what is and what isn't a robot, however a reasonable definition is:

“ *a mechanism which moves and reacts to its environment* ”

– <http://www.robotstoreuk.com/ROBOT%20PAGES/Robot%20definition-1.htm>, Sep 2012

Another definition that captures most people's idea of a robot is:

“ *a machine that functions in place of a living agent* ”

– <http://inventors.about.com/od/roboticsrobots/a/RobotDefinition.htm>, Sep 2012

Breaking the first definition into its two parts, it specifies that a robot is 1) capable of movement and 2) capable reacting to its environment.

Movement is generally achieved using some form of motor or actuator (the LEGO® NXT has three of them). Interaction with the environment is achieved using sensors. The type of sensors that a robot is equipped with depends on its purpose. A robot that needs to land on Mars would need to detect how far it is from the surface, so would probably be equipped with some form of distance sensor or radar. An assembly line robot on the other hand might be equipped with complex visual sensors, so that it can position screws and fittings correctly.

That covers the movement and sensing capabilities of a robot, but a sort of “brain” is required to tie everything together. After all, if our robot is functioning in place of a living agent, we need it to have the ability to make some decisions for itself.

The brain of a robot is quite simply a computer. It takes input from sensors, makes some decisions based on that input, and produces output in the form of motion or some other interaction with its environment. It's conceptually the same as the "brain" - or CPU - in your laptop computer or mobile phone, only rather than responding to keyboard strokes and mouse movement with information on the screen, a robot's brain responds to different sensory inputs with any imaginable type of output, be it mechanical, electrical or chemical.

Figure 1.2 The LEGO® NXT 2.0 Robot



© The LEGO® Group. All Rights Reserved.

Why use robots?

There's many reasons why we might want to use a robot in place of a human. Here's just a few, can you think of any more?

A robot can:

- Perform repetitive tasks without becoming bored or losing concentration.
- Perform dangerous tasks would normally risk the life of a human. For example:
 - Minefield mapping
 - Nuclear disaster cleanup
 - Bomb disposal
- Go places that are difficult for humans to survive in. For example:
 - Outer space
 - Ocean floor
 - A burning building
- Operate for extended periods of time without assistance – the [NASA Voyager](#) missions have been sending back data from the depths of our solar system for 33 years without any physical contact from humans!

Piazza [Discussion](#)

(see ["What is Piazza?"](#) to learn about this feature)

Section 2

Enchanting

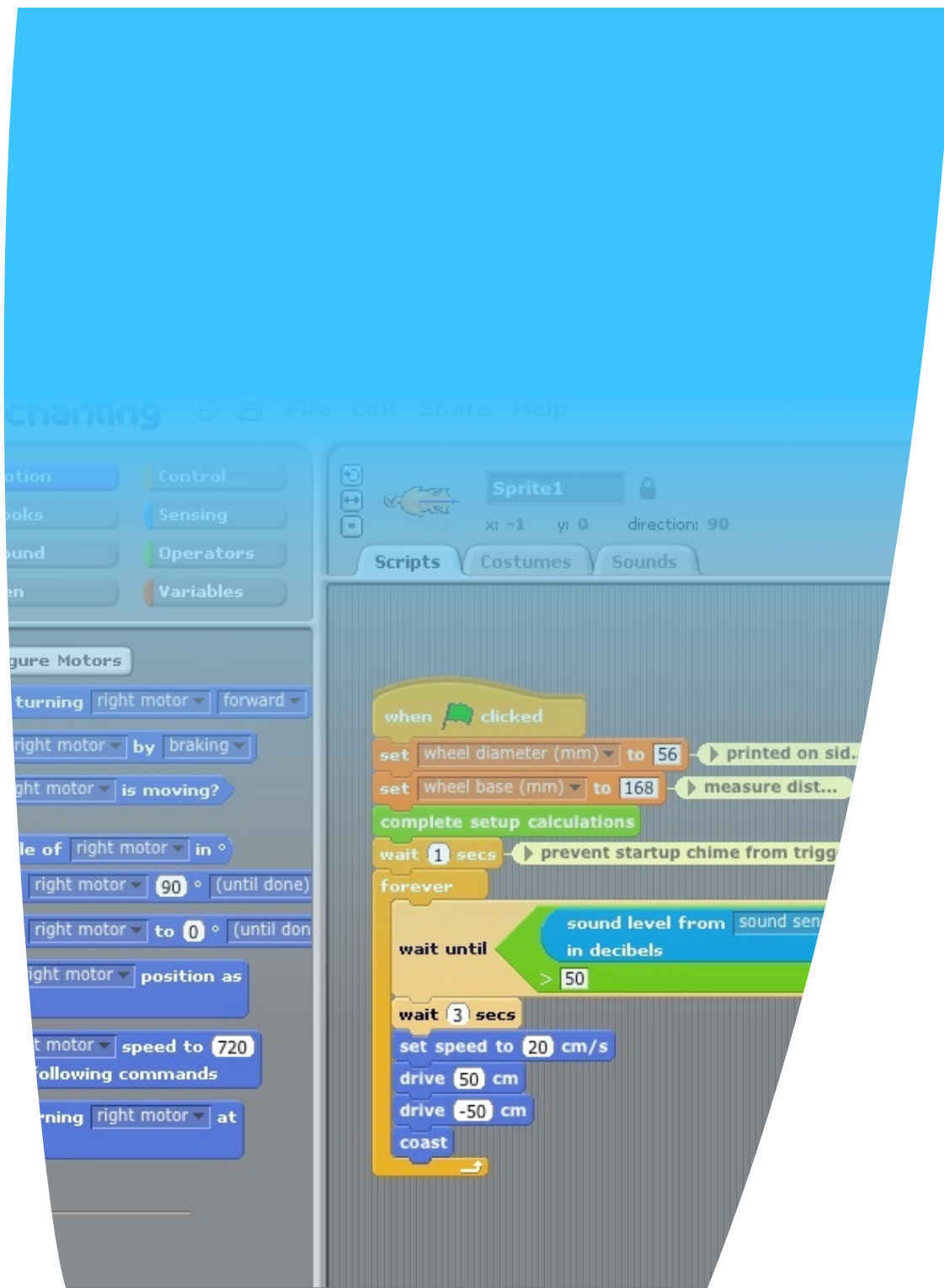
In this book you will learn how to program a LEGO® NXT robot using [Enchanting](#). Enchanting is an easy-to-use graphical programming tool, designed specifically for [LEGO® MINDSTORMS NXT](#) robots.

Along with [Scribble](#) and [Cellular](#), Enchanting is an open-source extension to the fantastic [BYOB](#) (Build Your Own Blocks) platform, which is developed by the [University of California, Berkley](#). Both Scribble and Cellular are developed by the [Monash University Faculty of Information Technology](#), while Enchanting is primarily developed by [SABRE](#) (Southern Alberta Robotics Enthusiasts).

For more information, please see [SNAP! – The Future of BYOB](#).

To get started, download Enchanting for *Mac OS X* and *Windows* from the link below and check out the example files included with the program.

<http://monash-blockbooks.appspot.com>

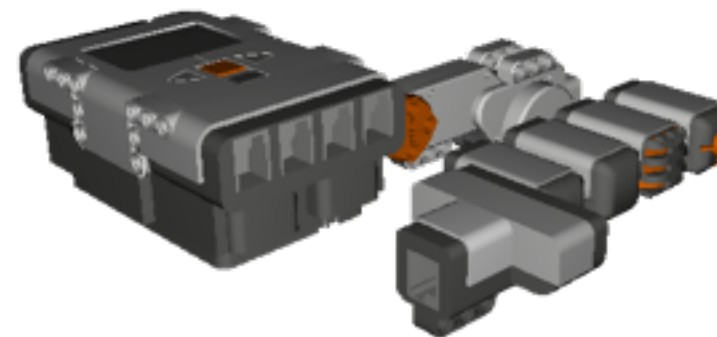


The LEGO® NXT Robot

In this section we'll briefly discuss the different electronic components of the LEGO® NXT and what they do.

Interactive 1.1 LEGO® NXT Components

*Spin me, I'm 3D!**



"Lego Mindstorms NXT Components"
Trimble 3D Warehouse SketchUp Model
© 2007 [paytonwhite](#). All Rights Reserved.

* Not available in the PDF version of this book.

Brain

The computer that controls the NXT is appropriately called a brain. It can print information on the screen, receive input from the sensors, send electricity to the motors to turn them, and allows us to communicate with it from Enchanting over USB or bluetooth. The NXT's brain performs a similar function to what yours does— it controls all the components (the equivalent of body parts) of the robot based on the sensory information available to it. When we write programs for the NXT, we are actually writing a set of instructions on how we'd like the brain to use its components under different conditions.

Video tutorial 1 demonstrates how to configure Enchanting to communicate with your NXT.

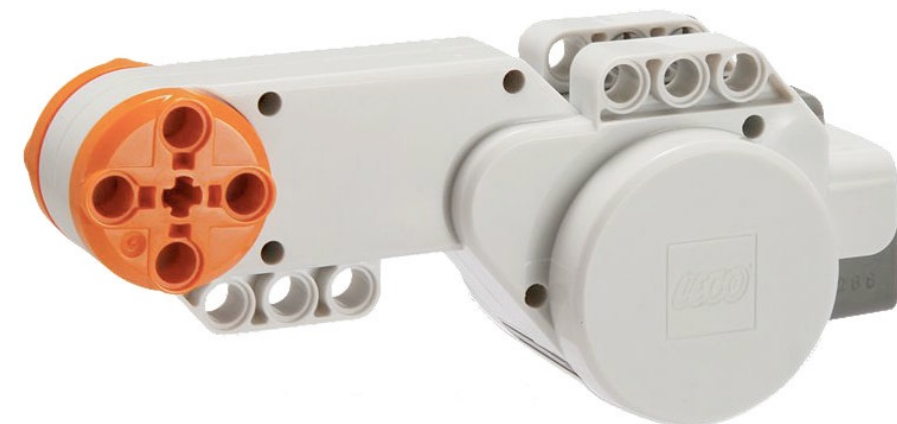
Interactive Servo Motors

The motors on the NXT not only provide the ability to drive the robot around like a car, but in different robot configurations can also act as servos to move different components around (even sensors), sort of like arms for your robot. In fact, the Interactive Servo Motor even provides feedback to the brain on its present rotation, so it can be used as a form of input (or sensor) too.



LEGO® NXT “Brain” (Intelligent Brick)

*Note the orange button is referred to as the enter key in Enchanting, not to be confused with the enter key on your computer keyboard.
© The LEGO® Group. All Rights Reserved.*



LEGO® NXT Interactive Servo Motor

© The LEGO® Group. All Rights Reserved.

Sensors

The NXT's sensors allow the robot to be aware of different aspects of its surrounding environment, in the same way that animals (including humans) use their sensory abilities for everyday decision making. Using information from the robot's sensors, we are able to program the brain to perform different actions based on its surrounding environment. Once we have programmed the robot, it is able to perform these actions at the appropriate time without intervention from a human – in other words it can operate autonomously.

Tutorial 3 demonstrates how to configure Enchanting to work with your different sensors.



Light Sensor
Measures Brightness



Touch Sensor
Detects Physical Contact



Sound Sensor
Measures Volume



Ultrasonic Sensor
Measures Distance

All images on this page are © The LEGO® Group. All Rights Reserved.

Section 4

Piazza

Throughout this book, you will find links to [Piazza](https://piazza.com). This is an online question and answer platform, which can be accessed via it's website at piazza.com or via the sleek [Piazza](https://piazza.com) iOS (iPhone and iPad) app.

What's Piazza?



Piazza is a free platform for instructors to efficiently manage class Q&A. Students can post questions and collaborate to edit responses to these questions. Instructors can also answer questions, endorse student answers, and edit or delete any posted content.

Piazza is designed to simulate real class discussion. It aims to get high quality answers to difficult questions, fast!

The name Piazza comes from the Italian word for plaza--a common city square where people can come together to share knowledge and ideas. We strive to recreate that communal atmosphere among students and instructors.

– <https://piazza.com/school-search>, October 2012

To use this service, you'll first need to join the Enchanting class on Piazza...

[Join the Enchanting class now!](#)

Chapter 2

Tutorials

Tutorials

In the sections that follow, you will be presented with a series of video tutorials, which will expose you to the fundamentals of programming your LEGO® NXT robot using [Enchanting](#) on your desktop computer. After **Getting to Know Your NXT Robot**, each tutorial follows with a set of hands-on activities and questions to try along the way. Upon completion of this chapter, you will be equipped with the skills to begin working on the more complex challenges later in the book.

For all of these activities, your NXT robot will need to be in a “standard car” configuration, as depicted in the tutorials. This is a 3-wheeled car, comprising of 2 drive motors (left and right) and a jockey wheel. Different activities will require different sensors to be connected to the robot.

Now let's begin!



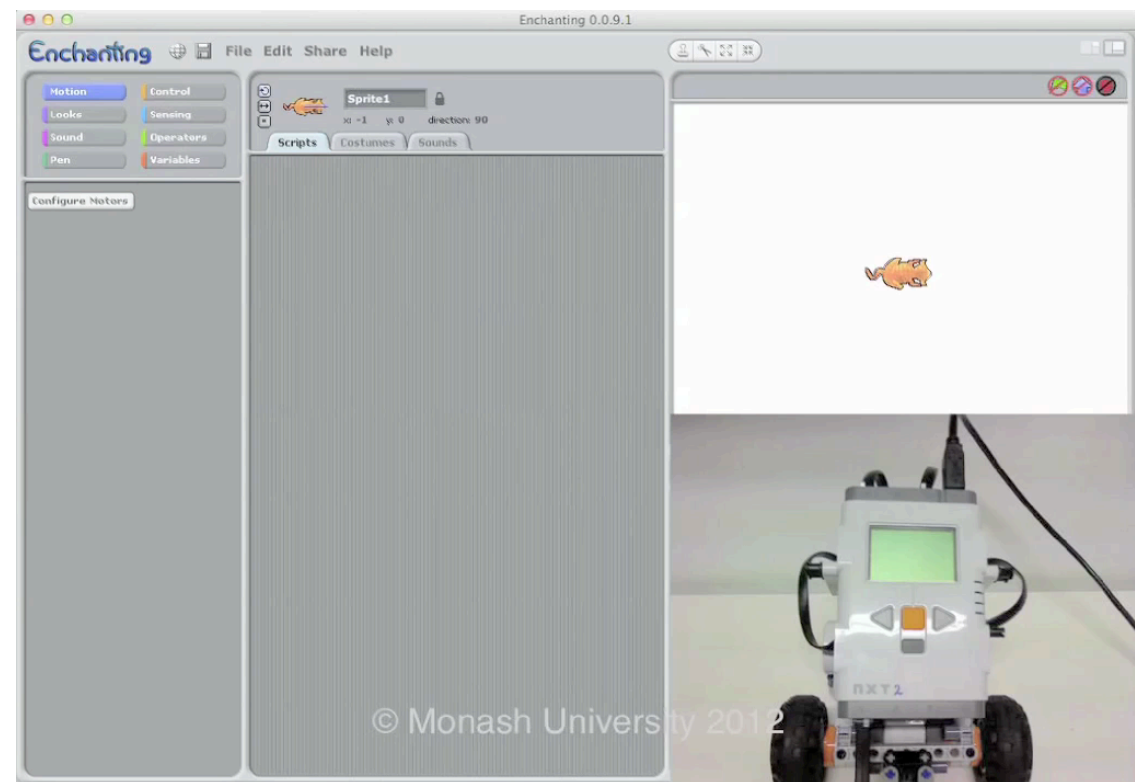
NXT Brain with Various Motors and Sensors Attached

© The LEGO® Group. All Rights Reserved.

Getting to Know Your NXT Robot

In this first tutorial, we'll look at the very basics of using Enchanting to program your LEGO® NXT robot. This includes how to connect to your robot and how to ensure that you have the correct firmware loaded onto it. You'll also learn how to run a basic program that plays some tones and prints some text to its screen.

Tutorial 2.1 The Basics – Using Enchanting to Program Your Robot

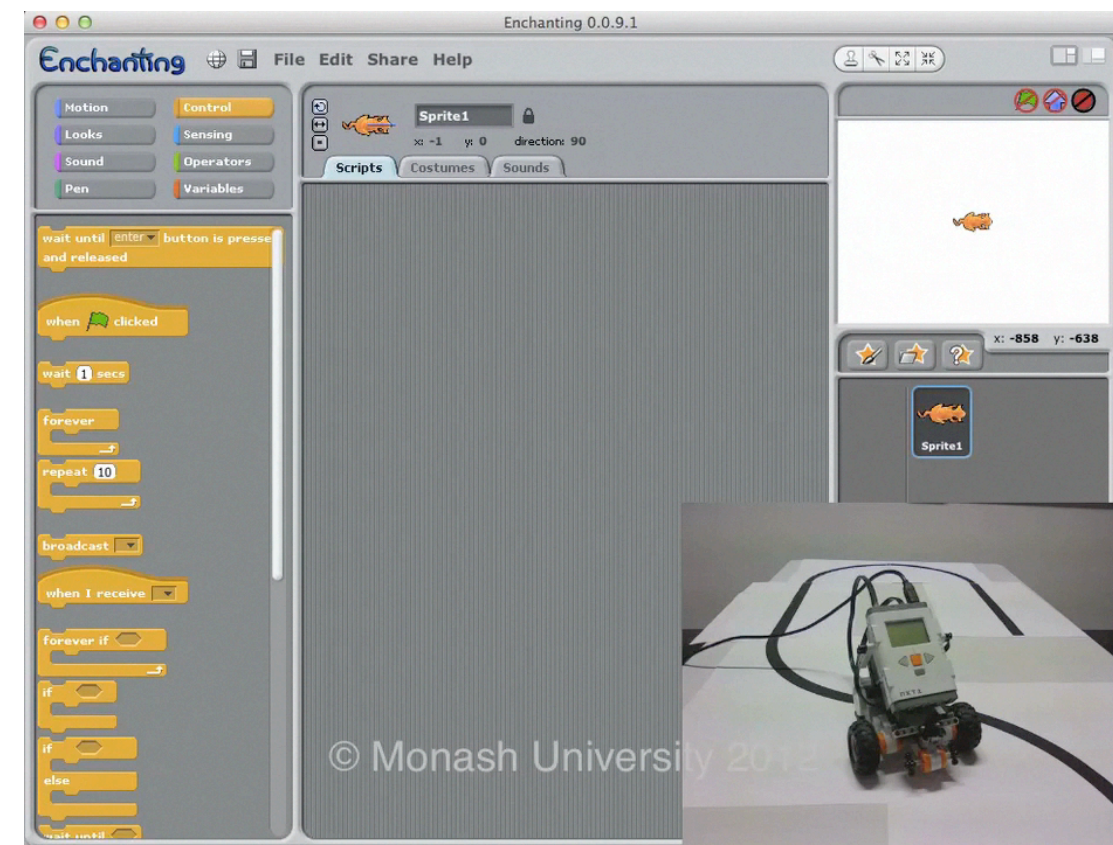
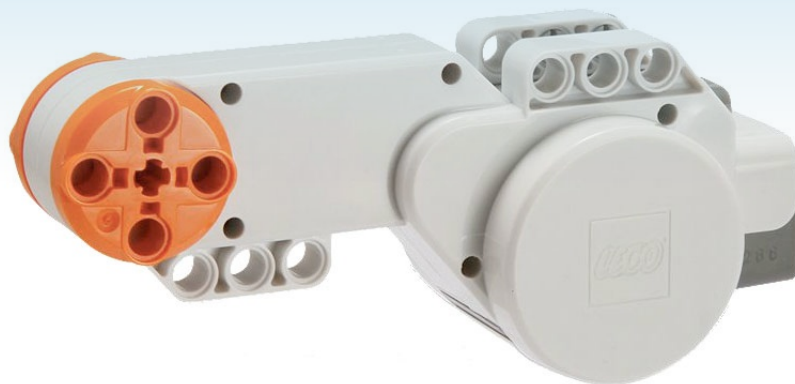


Tutorial 2

Motor Skills

In this tutorial you'll learn how to control your NXT robot's motors. This includes how to configure the motors for the “standard car configuration”, how to make it drive forward & backward, turn left & right and operate at different speeds.

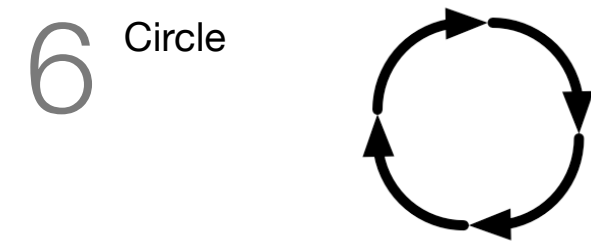
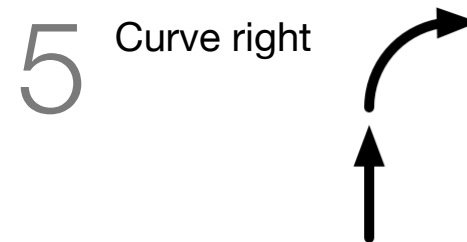
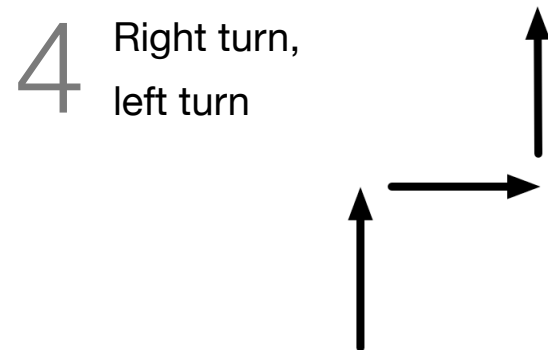
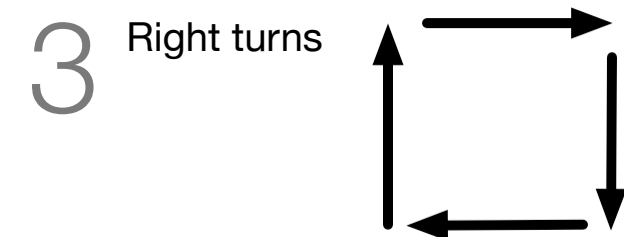
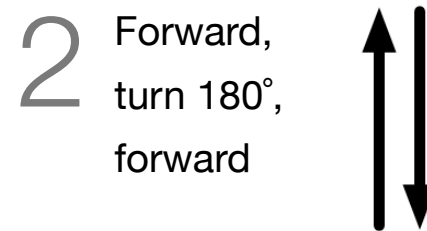
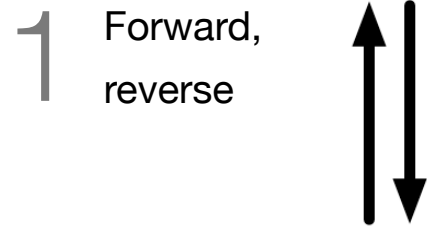
Tutorial 2.2 The Motors – Configuration, Driving, Turning and Speed



Activities & Questions

Activities

For each path below, program your robot to drive as indicated:



Piazza [Discussion](#) & [Solution](#)

(see [“What is Piazza?”](#) to learn about this feature)

Question

For **part 2**, by how many degrees should each motor rotate in order to turn the robot 180°? (assuming [“standard car”](#) configuration)

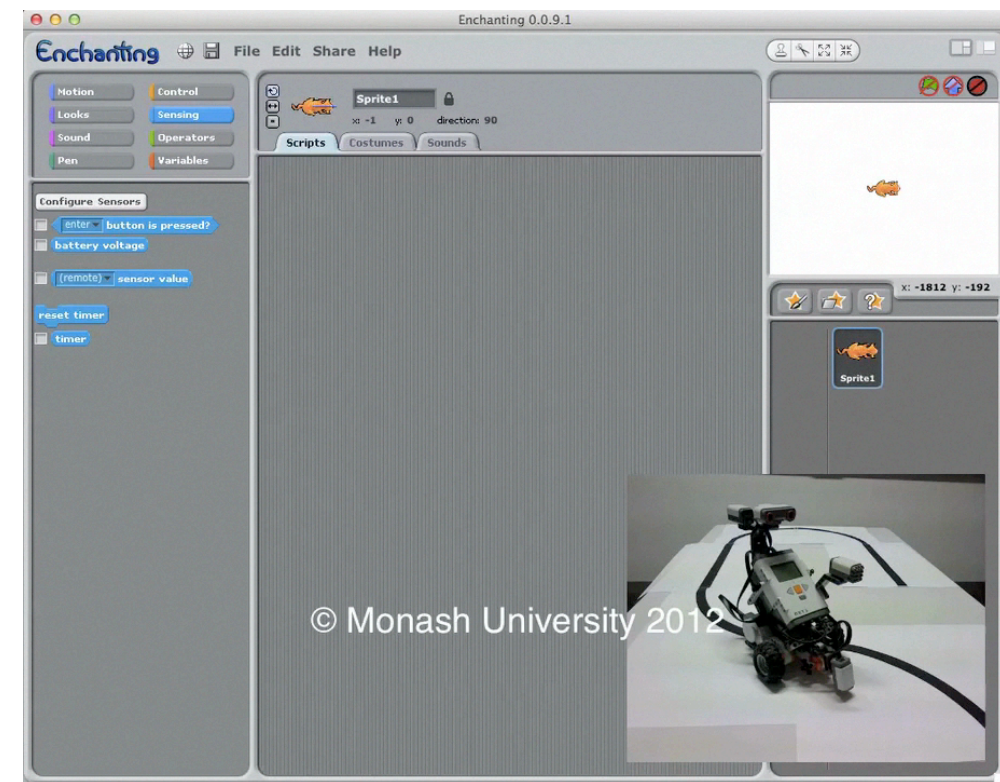
Answer Me

Piazza [Discussion](#) & [Solution](#)

Touch Sensor

In this tutorial you'll be introduced to each of the NXT robot's sensors (touch, ultrasonic, sound and light) and learn how to configure Enchanting to use each of them. You'll then learn how to use the touch sensor to detect and respond to physical contact, such as bumping into an obstacle.

Tutorial 2.3 The Sensors – Configuration and Detecting Touch



Activities & Questions

Activities

Complete the following activities with a touch sensor attached to your robot:

1. Drive forward, stop when the touch sensor is pressed.
2. Drive (forward / backward), change direction to (backward / forward) when touch sensor pressed (repeat behaviour continuously).
3. Drive forward, turn around when the touch sensor is pressed, continue driving (repeat behaviour continuously).
4. Wait for the touch sensor to be pressed, drive backward ~10cm (repeat behaviour continuously).

Piazza [Discussion](#) & [Solution](#)

Question

For **part 2**, what statement would you use to decide when the robot should change direction?

Answer Me

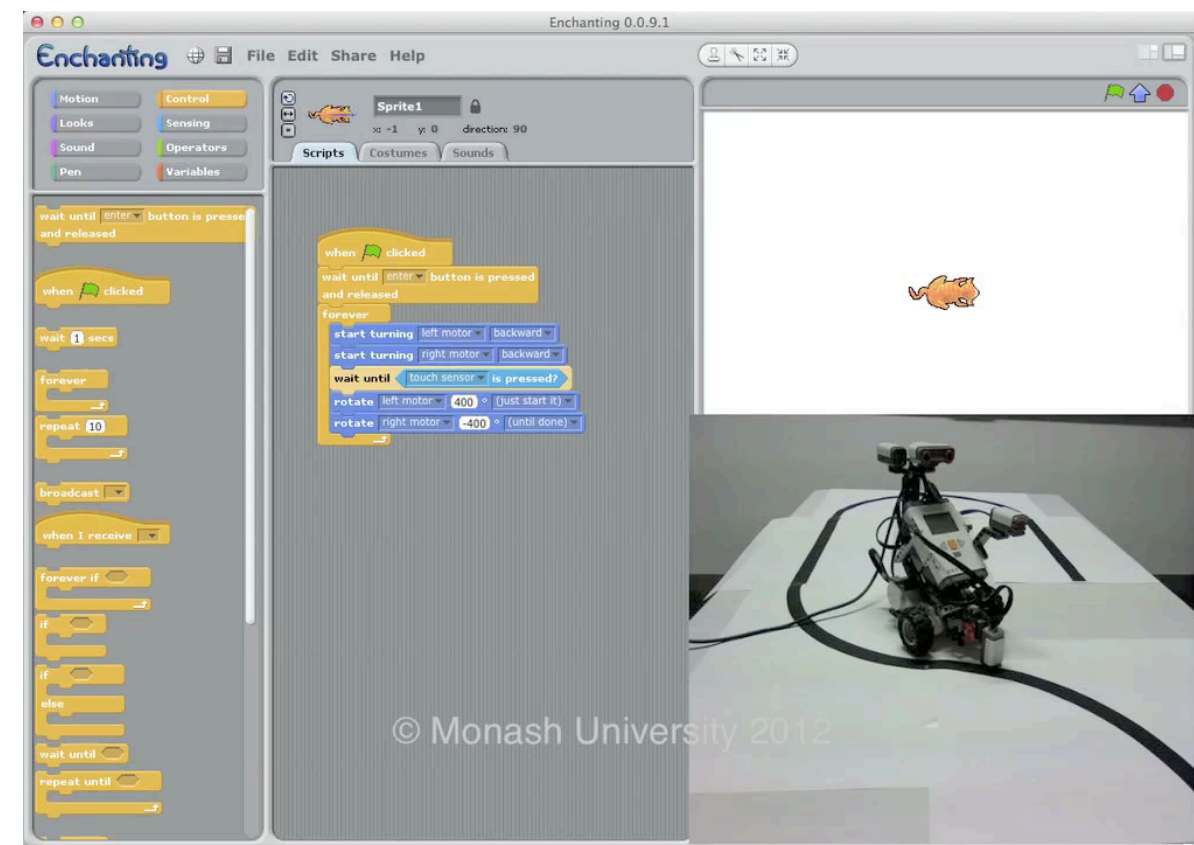
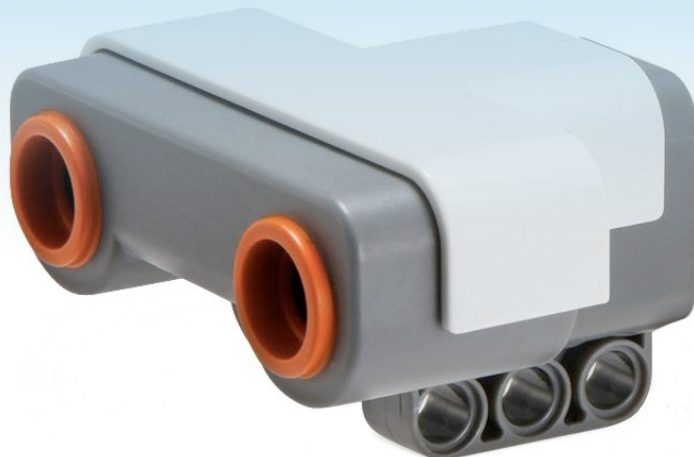
Piazza [Discussion](#) & [Solution](#)

Tutorial 4

Ultrasonic Sensor

In this tutorial you'll learn how to use the [ultrasonic sensor](#) to determine the distance between the NXT robot (or this sensor, to be precise) and an object. You'll then learn how to use this sensor to enable the robot to “chase” something.

Tutorial 2.4 The Ultrasonic Sensor – Measuring Distance to Objects



Activities & Questions

Activities

Complete the following activities with both the touch sensor and the ultrasonic sensor connected to your robot:

1. Drive forward, stop when there is an obstacle within 20 cm of the robot.
2. Drive forward, turn around when there is an obstacle within 20 cm of the robot, continue driving (repeat behaviour continuously).
3. Wait until the touch sensor is pressed, then drive forward until an obstacle is within 20 cm of the robot.
4. Drive forward until either the touch sensor is pressed OR the robot is within 20 cm of an obstacle, then turn to a new random direction and continue driving (repeat behaviour continuously).

Piazza [Discussion](#) & [Solution](#)

Questions

- a. For **part 2**, what statement would you use to decide when the robot should turn around?

Answer Me

Piazza [Discussion](#) & [Solution](#)

- b. For **part 4**, how do you turn the robot to a new random direction? (Hint: you will need to use a “variable”)

Answer Me

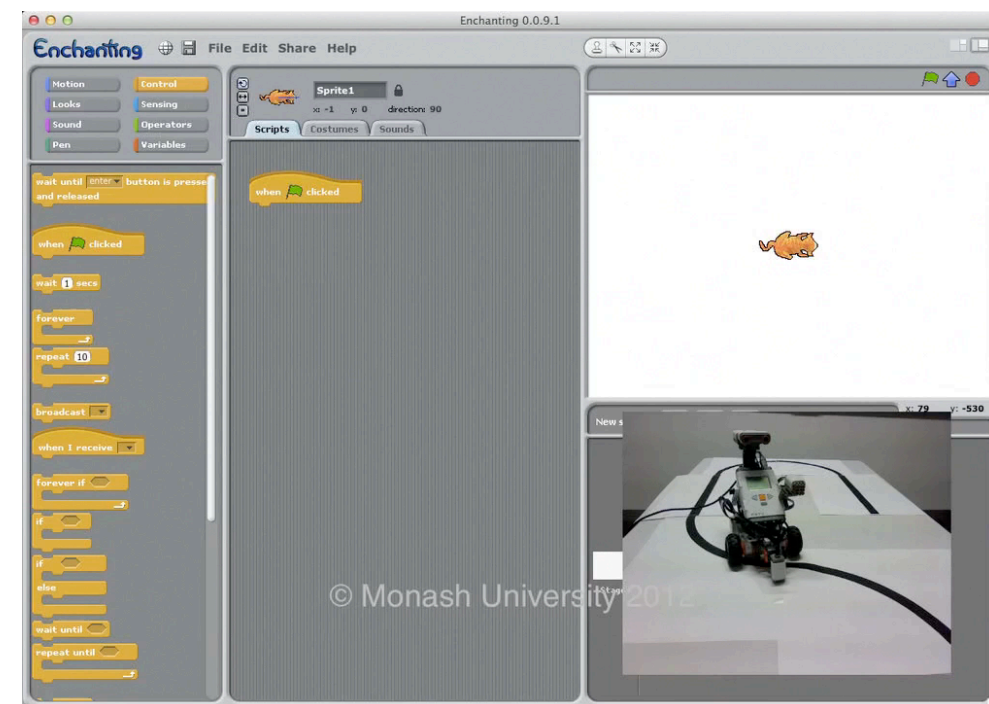
Piazza [Discussion](#) & [Solution](#)

Sound Sensor



In this tutorial you'll learn how to use the sound sensor (essentially a microphone) to determine how loud the environment is and how to get your NXT robot to respond to noises, such as the clicking of fingers. There's also a tip on how to synchronise the start time of two or more robots, as needed in some of the challenges, such as Tug of War and Robot Sumo.

Tutorial 2.5 The Sound Sensor – Measuring Loudness of Noises



Activities & Questions

Activities

Attach the sound sensor to your robot for the following activities.

1. Wait until the robot detects a loud noise (a hand clap), then start driving forward.
2. Drive forward, when the robot detects a loud noise turn 180° and continue driving (repeat behaviour continuously).
3. Drive forward, continuously setting the wheel speed to the measured volume.

Piazza [Discussion](#) & [Solution](#)

Questions

- a. For **part 2**, what statement would you use to decide when the robot should turn 180°?

Answer Me

Piazza [Discussion](#) & [Solution](#)

- b. For **part 3**, what statement would you use to turn the left motor?

Answer Me

Piazza [Discussion](#) & [Solution](#)

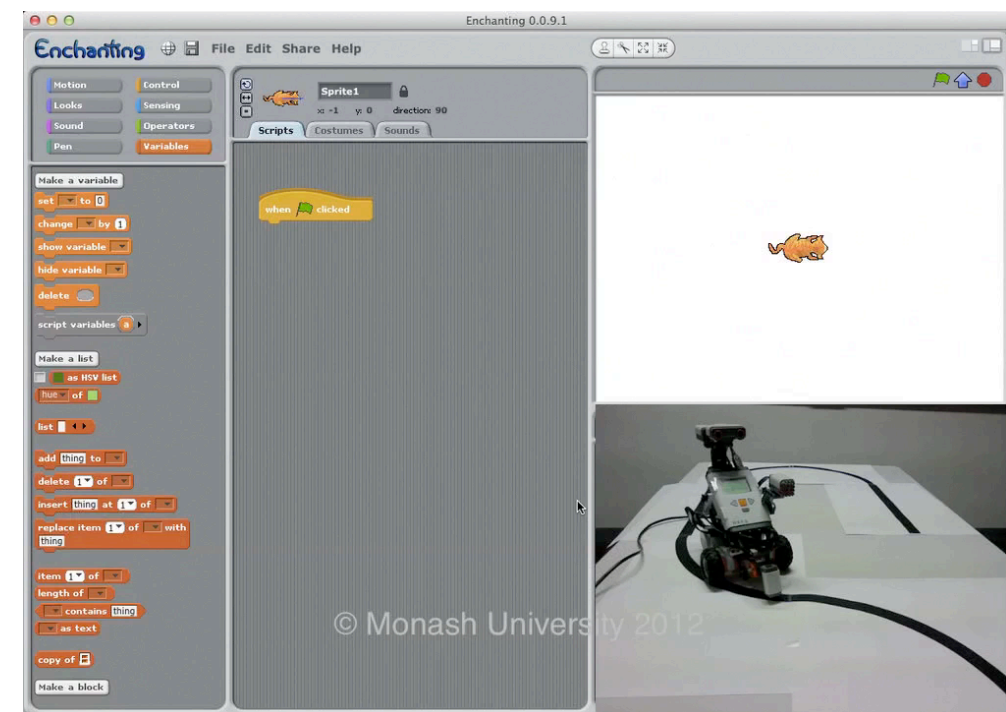
Tutorial 6

Light Sensor



In this tutorial you'll learn how to use the light sensor to take readings of brightness. In this case, it'll be the lightness or darkness of the surface below the NXT robot. You'll also learn how to make the robot **auto-calibrate** itself to handle different lighting conditions. This will then be used to demonstrate how to keep the robot moving, while being constrained to a particular area.

Tutorial 2.6 The Light Sensor – Measuring Brightness of Surfaces



Activities & Questions

Activities

Attach the light sensor to your robot for the following activities.

The light sensor should be approximately 1 cm from the surface your robot is sitting on.

Remember that you will need to auto-calibrate the light sensor as part of your program (see **Auto-Calibration**).

So long as you run the auto-calibration every time the program starts, your program can be reused in different lighting conditions without modification!

1. Drive forward until a black line is detected, then stop.
2. Drive forward until a black line is detected, then turn around and continue driving (repeat behaviour continuously).
3. Drive quickly when over a bright surface, slowly when over a dark surface.
4. Drive forward, print the brightness level to the screen every second (repeat behaviour continuously).

Piazza [Discussion](#) & [Solution](#)

Question

For **part 2**, what statement would you use to decide when the robot should turn around?

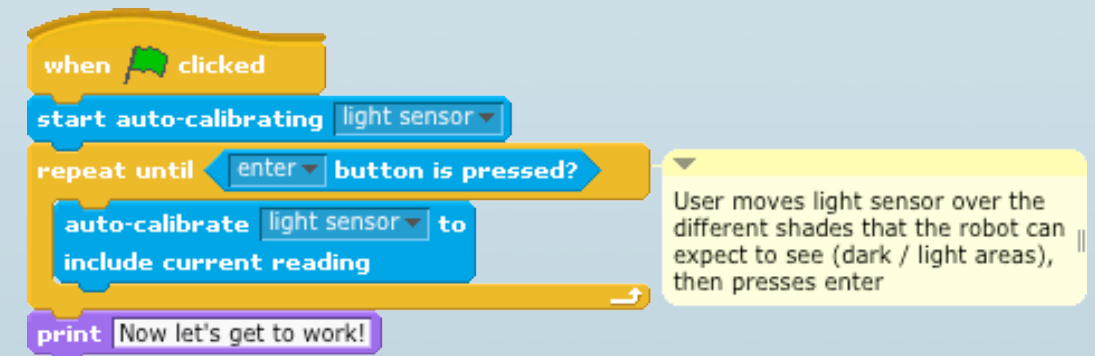
Answer Me

Piazza [Discussion](#) & [Solution](#)

Auto-Calibration

Here's an example of an auto-calibration technique that can be used at the start of a program:

Figure 2.1 Auto-Calibration Example



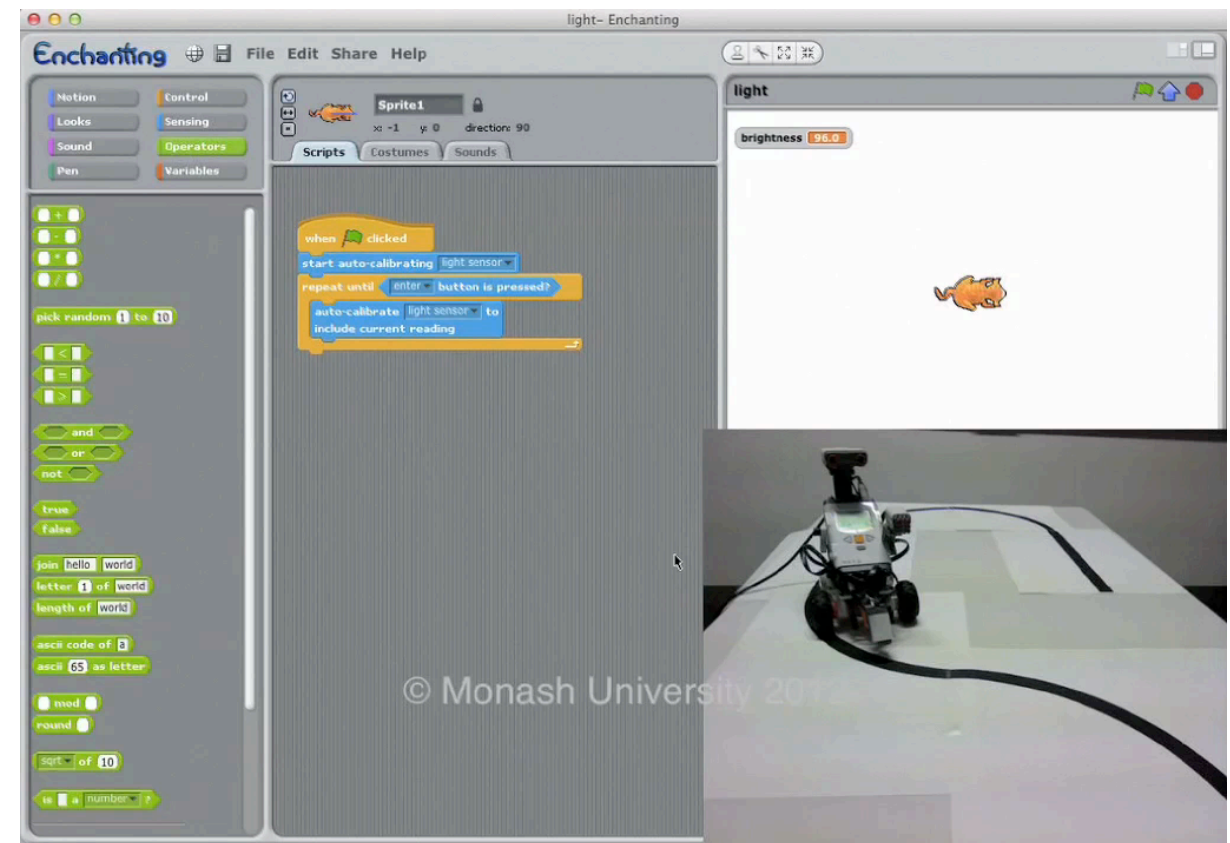
The “raw” brightness level from the LEGO® Light Sensor varies depending on the lighting conditions you are working with. This can make it difficult to use your programs in different lighting conditions, for example moving from inside to outside.

Auto-calibration allows you to show the robot the different brightness levels it can expect to see in its current conditions. It will then translate the raw brightness values such that the darkest area it was shown produces 0, and the brightest produces 100. All the shades in between will fall somewhere between 0 and 100.

Line Follower

In this final tutorial, we'll extend upon **Tutorial 6** to use the light sensor to enable your NXT robot to follow the edge of a black line. The line itself may be straight, curved, in the shape of a racing track or anything else—be creative!

Tutorial 2.7 The Line Follower – Following the Edge of a Black Line



Activities & Questions

Line following is an important part of your programming toolbox when using the LEGO® NXT. It allows you to create paths through the environment for your robot to follow, meaning that your robot can perform tasks that require basic navigation. There are many different ways of creating a line follower depending on how many light sensors you have available, however in this activity it's assumed that you only have one.

Figure 2.2 Following a Black Line



Let's say we want to follow a black line, like the one pictured above. You might think that the way to achieve this would be to ensure that the robot's light sensor is always directly over a black surface.

However, there's a problem with this strategy:

Question A

What happens when we detect the white surface?
How do we know if we've veered off to the left or the right?

[*Reveal solution on Piazza*](#)

• •

Piazza [*Discussion*](#)

The solution is to instead follow the *edge* of the black line, rather than the line itself. Either edge is fine, we'll use the lower edge in this example.

Figure 2.3 Following the *Edge* of a Line



If we have correctly **auto-calibrated** the light sensor, then we should read a value of roughly 0 when over the black surface, and 100 when over the white. The reading on the edge of the black line, when the sensor is positioned over half black and half white, should be 50.

Question B

How do we keep the robot positioned over the edge of the line as it moves forward?

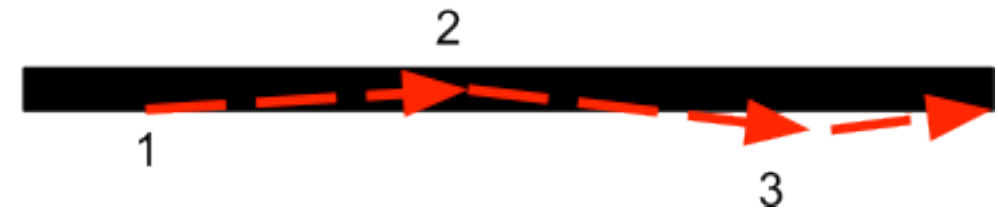
[Reveal solution on Piazza](#)

• •

Piazza [Discussion](#)

Essentially what we need to do is set the speed of the right wheel relative to the **brightness**, and the left wheel relative to the **darkness**. In doing so, the robot will always settle over the edge of the line.

Figure 2.4 Steps to Following the Edge of a Line



In the example above, the sensor starts in the correct position over the edge of the line (1). As the robot moves forward, it veers too far left (2), causing the left wheel speed to be increased while the right wheel speed is decreased. By the time the robot has reached (3), it has overcompensated and now the reverse occurs.

So long as the robot is not moving too fast, it will eventually stabilise directly over the bottom edge of the line.

To achieve this behaviour, we identified that the right wheel must be set relative to the brightness, and the left wheel must be set relative to the darkness. The light sensor **ONLY** provides a measurement of brightness, so we need to reverse the values that it produces in order to get a measurement of darkness.

Question C

If our light sensor produces values between 0 and 100, how can we convert it to darkness?

i.e. 0 brightness = 100 darkness
20 brightness = 80 darkness
50 brightness = 50 darkness
... and so on

[Reveal solution on Piazza](#)

• •

[Piazza Discussion](#)

Wheel Speed Calculation Table (for Right-Hand Edge)

	Left wheel	Right wheel
Moves relative to	darkness	brightness
Speed calculation	100 – sensor brightness	sensor brightness

Activities

Using the calculations from this table, create a line following program. Modify it to achieve each of these tasks:

1. Follow the RH (right hand) edge of a black line (according to the given example).
2. Follow the LH edge of a black line.
3. Move along the LH of the black line 3 times as quickly.
4. Follow a line until a loud noise is detected, then turn around 180° and follow in the opposite direction (still on LH edge).

[Piazza Discussion](#) & [Solution](#)

Further Questions

- d. For **part 1**, what calculation should the “start turning *right motor* at” block use?

[Answer Me](#)

[Piazza Discussion](#) & [Solution](#)

- e. For **part 3**, what calculation should the “start turning *right motor* at” block use?

[Answer Me](#)

[Piazza Discussion](#) & [Solution](#)

Chapter 3

Challenges

Challenge 1

Tug of War

Requirements

Robots: 2

Sensors: *Sound sensor*

Props: *Length of string,
Electrical tape*

Feedback

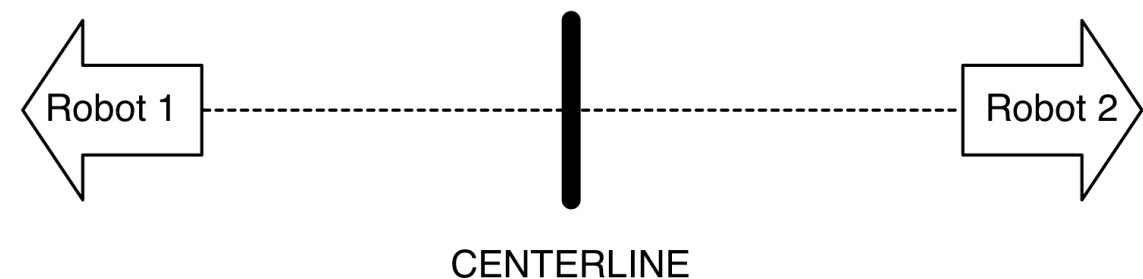
Challenge

Your robot is competing in a tug-of-war match. Create a program to try to beat your opponent.

Setup

Two robots are positioned facing each other with a length of string connecting them. Mark a centreline with electrical on the playing surface, midway between the two robots.

Figure 3.1 The Field



Rules

- A robot must not move until it detects a loud noise (handclap) indicating the start of the match.
- A robot may only move forward and backward (no turning on the spot).
- If a robot is pulled over the centreline, it loses.
- Players decide on best of 3, best of 5, etc.

Ideas

- Experiment with different motor speeds.
- Experiment with starting / stopping the motors periodically.

Piazza [Discussion](#)



Adaptation of [“Tug of War”](#)
© 2006 [toffehoff](#), [CC BY-SA 2.0](#)

Challenge 2

Velociraptor

Requirements

Robots: 1

Sensors: *Light sensor*

Props: *White playing surface,
Black electrical tape*

Feedback

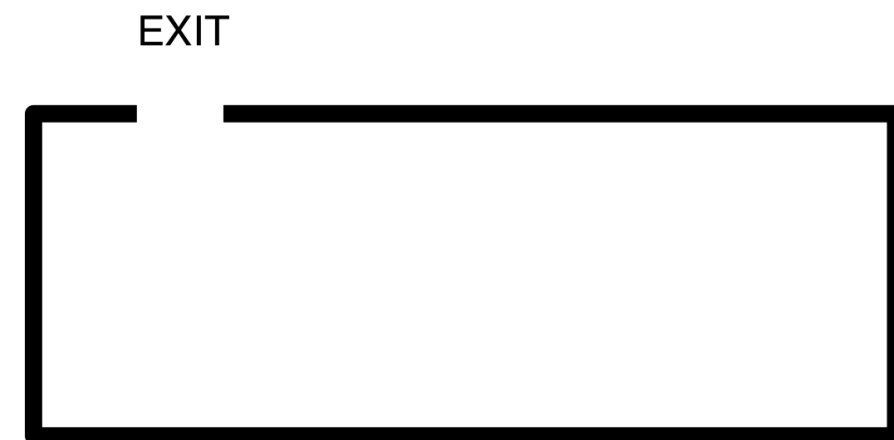
Challenge

Your Velociraptor (robot) needs to find the gap in the fence to escape. Write a program to efficiently find the gap in the fence, and have your robot drive through it to freedom.

Setup

Mark out a large dinosaur enclosure using the electrical tape to form a “fence”. It should completely enclose the robot, except for one gap, equal to the width of your robot.

Figure 3.2 The Velociraptor’s Enclosure



Rules

- The robot must not cross over any black line.
- The robot must eventually find the gap in the fence and escape.

Ideas

- Experiment with different approaches, have the robot try different areas randomly, then try to improve it with a more methodical approach.
- Time your different approaches and work out which one is the best.

Piazza [Discussion](#)



Adaptation of "[Velociraptor!](#)"
© 2011 [Eric Kilby](#), [CC BY-SA 2.0](#)

Challenge 3

Robot Sumo

Requirements

Robots: 2

Sensors: *Light sensor,
Sound sensor*

Props: *Sumo arena
(large dark circle on white paper)*

Feedback

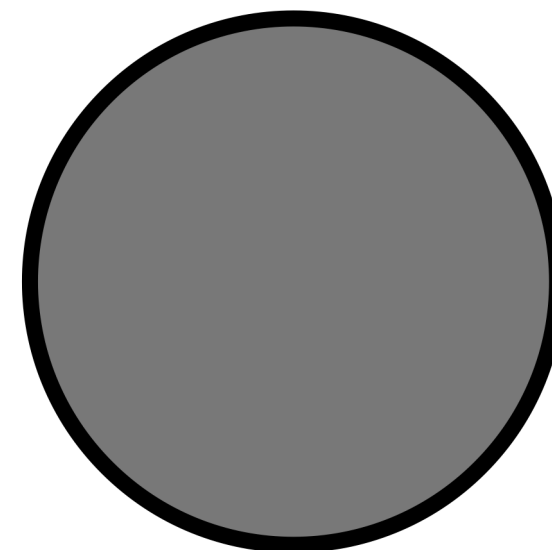
Challenge

Your robots are competing in a full contact sumo battle. The first robot to leave the sumo circle loses, either by driving over the boundary or being pushed by the other robot.

Setup

Place 2 robots facing each other at the centre of the sumo arena (a large dark circle on white paper).

Figure 3.3 The Sumo Arena



Adaptation of "[Jocelyn Foye - Sumo - Face Off - Gateway Japan @ Torrance Art Museum \(small\)](#)"

© 2011 Marshall Astor, CC BY-SA 2.0

Rules

- Robots must wait for loud noise (handclap) before moving.
- If a robot's wheels exit the sumo circle it loses.
- Both robots leaving the circle results in a tie.
- Any strategy for pushing the other robot out of the circle may be used, including the use of all available sensors.
- Participants decide on best of 3, best of 5, etc.

Ideas

- Ensure your robot uses its light sensor to stay within the sumo circle at all times.
- Experiment with different motor speeds, forward, reverse, etc.
- Try to make use of other sensors (touch, ultrasonic) to react to your opponents moves.



Piazza [Discussion](#)

Adaptation of "[Jocelyn Foye - Sumo - Face Off - Gateway Japan @ Torrance Art Museum \(small\)](#)"
© 2011 [Marshall Astor](#), [CC BY-SA 2.0](#)

Challenge 4

Bomb Squad

Requirements

Robots: 1

Sensors: *Light sensor,
Ultrasonic sensor
or Touch sensor*

Props: *Electrical tape,
Small cardboard box to
represent bomb
(or other moveable object)*

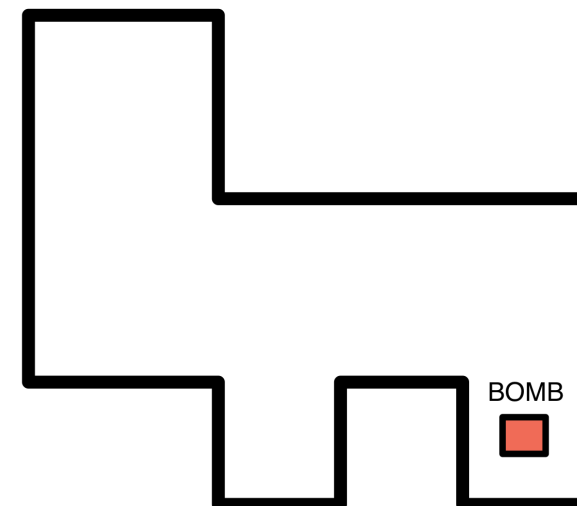
Challenge

Your bomb removal robot has been deployed in a building containing a bomb. It needs to locate the bomb, and push it outside of the building before sounding an “all clear”.

Setup

Your bomb should be tall enough that it can be detected by the robot’s ultrasonic or touch sensor. Mark out the perimeter of a building– it can be any shape you like. Place the bomb somewhere in the building.

Figure 3.4 The Building and the Bomb



Rules

- The robot must stay within the building until the bomb has been removed
- The robot must locate and push the bomb over the perimeter line of the building
- When the robot detects that it has pushed the bomb out of the building (the robot's light sensor detects a perimeter wall AND the ultrasonic / touch sensor detects the bomb in front of it), it should sound an "all clear" by way of some audio tones.

Ideas

- Use the light sensor to stay within the perimeter.
- Use either the ultrasonic or touch sensor to detect contact with the bomb– if you are using the touch sensor you may need to use a heavier object to represent the bomb.



Piazza [Discussion](#)

Adaptation of ["Robot using its pigstick to disable an IED \(bomb\)"](#)
© 2007 [Alexandre Dulaunoy](#), [CC BY-SA 2.0](#)

Challenge 5

Maze Solver

Requirements

Robots: 1

Sensors: *Light sensor*
or Ultrasonic sensor

Props: *Electrical tape*
or Physical walls

Feedback

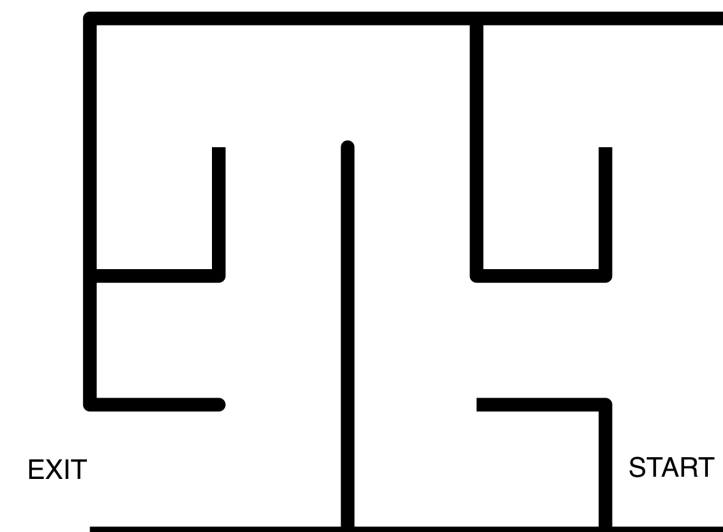
Challenge

Develop a strategy for your robot to find the exit to the maze.

Setup

Construct a maze using electrical tape or physical walls, the starting point can be anywhere in the maze, however there should be only one opening in an external wall for the robot to exit. The passageways in the maze should be wide enough to ensure your robot can move through them and turn corners without hitting the sides.

Figure 3.5 The Maze



Rules

- The robot must not cross any walls.
- The robot must successfully navigate to the exit of the maze.

Ideas

- Have a look at the [Random Mouse Algorithm on Wikipedia](#) for some ideas on maze solving algorithms.
- Start with the “random mouse” algorithm and then attempt some more difficult ones.

Piazza [Discussion](#)



Adaptation of [“Get Lost”](#)
© 2011 [Tim Green](#), [CC BY 2.0](#)

Challenge 6

Drag Race

Requirements

Robots: 2 (or more)

Sensors: Light sensor,
Sound sensor

Props: Electrical tape

Feedback

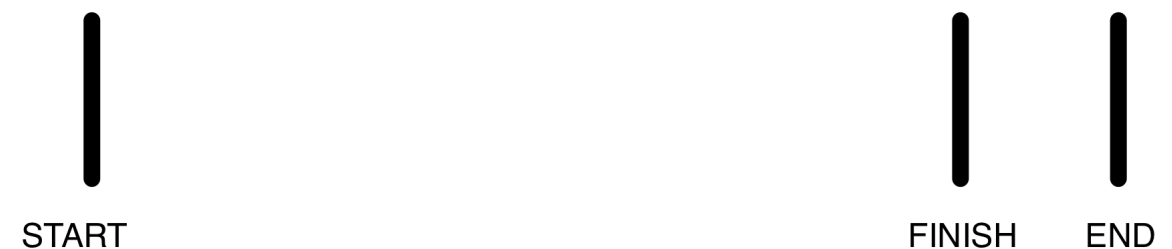
Challenge

Your robots are competing in a drag racing competition. Opponents start with their wheels behind the start line, and race toward the finish line after the starting sound. The winner is the drag racer which reaches the finish line first AND does not cross the end of the track.

Setup

Prepare the drag racing track by marking the start and finish points with electrical tape. Roughly 30 cm after the finish line, mark another line with electrical tape which represents the end of the track.

Figure 3.6 The Race Track



Rules

- Racers must wait for the starting sound before moving.
- The first racer to cross the finish line without going past the end line is declared the winner.
- Opponents decide on best-of-three, best-of-five etc.

Ideas

- Your drag racer will need to be able to detect the finish line (light sensor) to know when to stop.
- If your drag racer keeps going past the end of the track, you will need to experiment with different speeds. Try starting quickly, and then gradually getting slower so that you can stop quickly at the finishing line.

Piazza [Discussion](#)



Adaptation of "[Winston Drag Racing 1996](#)"
© 2008 [Marion Doss](#), [CC BY-SA 2.0](#)

Challenge 7

400m Finals

Requirements

Robots: 1

Sensors: *Light sensor,
Sound sensor*

Props: *Electrical tape*

Feedback

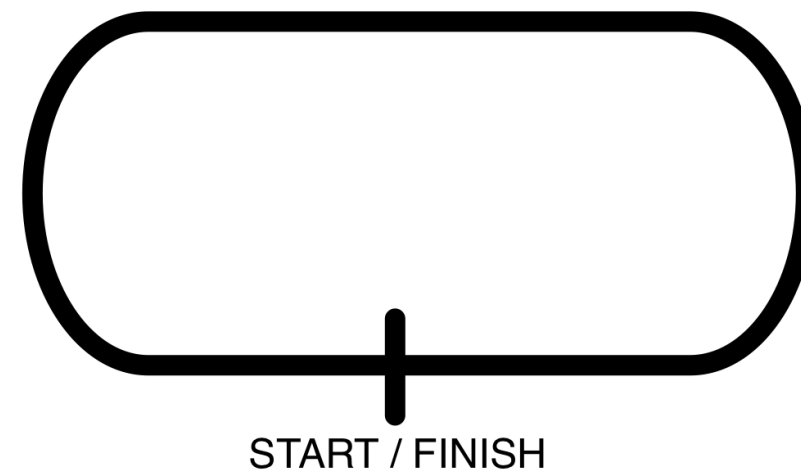
Challenge

Your robot is competing in the 400 meter track event at the Olympics. Create a line following program that drives the robot as quickly as possible around the circuit.

Setup

Mark out a large oval using electrical tape that represents an athletics track (2 straight sections, 2 curved sections). Use another small piece of tape to mark the start / finish line.

Figure 3.7 The Athletics Track



Rules

- The robot must not begin moving until the starting sound is heard.
- If there are multiple participants, time each robot to determine the winner.

Ideas

- The crucial factor is how fast your robot can drive around each bend. Too fast and it will be unable to stay on the line.

Piazza Discussion



Adaptation of "ISTAF Berlin 2010"
© 2010 André Zehetbauer, CC BY-SA 2.0

Challenge 8

The Mountaineer

Requirements

Robots: 1

Sensors: *Light sensor*

Props: *12 squares of paper in different shades of grey*

Challenge

Your robot is attempting to find the highest point on a mountain range. To do so, it must move over each area of the mountain, before returning to the highest point it found.


Setup

Prepare a pseudo-map of a mountain, where the different square tiles represent different altitudes of the terrain. Start your robot at lowest (lightest coloured) point on the terrain.

Figure 3.8 The “Mountain”



Auto-calibration of the light sensor is NOT allowed! You must use the raw brightness level from the sensor. The robot should not know which map tile is the darkest until it has traversed ALL of them!

You will still use the  block, just don't auto-calibrate at the start of your program.

Rules

- The robot must traverse every map tile.
- After traversing every map tile, the robot must return to the highest (darkest) point on the map.

Ideas

- Does your robot need to remember the height of every map location? Or does it just need to remember the location of the highest point?
- Think about how you will be able to reliably return to the highest point on the map.



Piazza [Discussion](#)

Adaptation of ["Mt. Shasta, Kevin, Dave, and Darcy"](#)
© 2005 [Darcy McCarty](#), [CC BY 2.0](#)

Appendix

SNAP! – The Future of BYOB

SNAP! – The Future of BYOB

What's BYOB?

“ *BYOB (Build Your Own Blocks) [is] an advanced offshoot of [Scratch](#), a visual programming language primarily for kids from the Lifelong Kindergarten Group at the MIT Media Lab. This version, developed by Jens Mönig with design input and documentation from Brian Harvey, is an attempt to extend the brilliant accessibility of Scratch to somewhat older users—in particular, non-CS-major computer science students—without becoming inaccessible to its original audience. BYOB 3 adds first class lists, sprites, and procedures to BYOB's original contribution of custom blocks and recursion.*

– <http://byob.berkeley.edu>, September 2012

This platform is developed by the [University of California, Berkley](#) and forms the basis of the [Scribble](#), [Cellular](#) and [Enchanting](#) applications. Both Scribble and Cellular are developed by the [Monash University Faculty of Information Technology](#), while Enchanting is primarily developed by [SABRE](#) (Southern Alberta Robotics Enthusiasts).

What's SNAP! ?

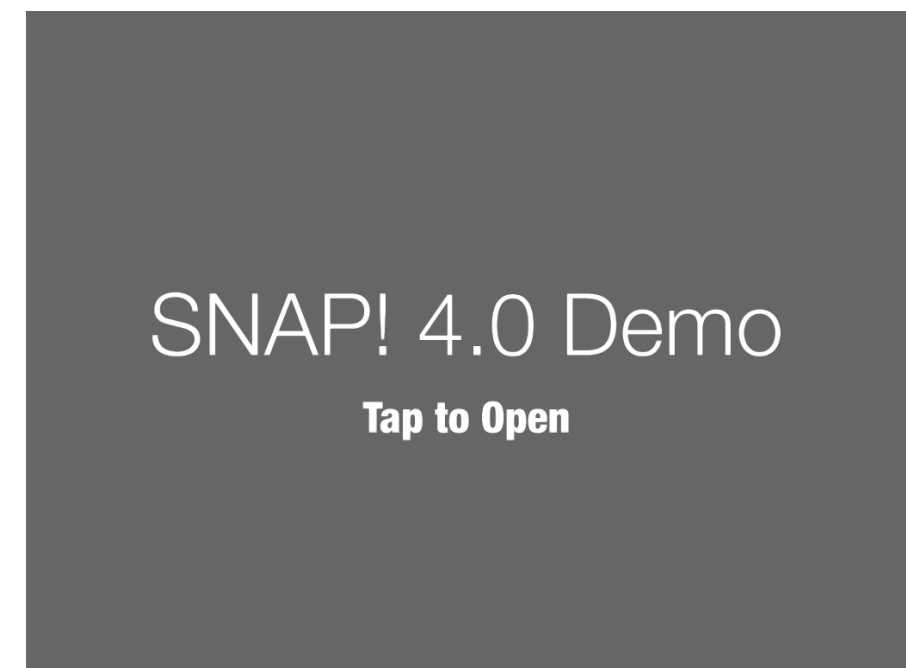
SNAP! is the future of BYOB. It represents a transition from a traditional desktop application to a “HTML” web application. This is exciting news because once this transition is complete, not only will you be able to program BYOB-style within your web browser, but also directly within eBooks, like as this one!

See <http://byob.berkeley.edu/> for more information.

Here's a sneak peak...

Note: This is an *online* preview – it requires internet access and may take a while to load.

Interactive Appendix.1 SNAP! 4.0 Alpha Demo



[Snap! 4.0](#) © 2012 Brian Harvey and Jens Mönig, available under the GNU Affero General Public License.