Preface

There is a growing recognition that Computational Thinking Practices are critical for all students to learn. They form the cornerstone of the language of innovation, and will drive all future STEM discoveries. They are a new set of “basic skills” that all students need to know.

But what are they? At first glance, concepts like “consider problems analytically” and “use data to inform decisions” seem abstract and difficult to comprehend. Educational robotics systems like the EV3 provide a much-needed tool to make them real and approachable.

Consider the first few activities in this curriculum: students program a robot to drive fixed distances in set patterns. Even these simple programming constructs require precise, thoughtful communication between the student and the robot – how far should the robot move? How far should it turn? As the challenges become more complex, students learn to break the large problems down into simpler ones, and construct solutions with care, one step at a time.

Sensors add the element of data and make key information about the robot’s environment available; numeric abstractions become a reality – 35 centimeters to the nearest wall, turn 90 degrees – and enable the student to make smart decisions about the robot’s behavior.

These practices – precise logical thinking, using data to make decisions, analyzing problems, and building solutions in teams – are critical in all forms of problem-solving, not just robotic ones.

Robotics activities are concrete, contextualized, and provide immediate feedback – important factors in satisfying a student’s desire for success and creating the motivation to continue learning. Students also learn about the robotics technologies themselves, which impact all modern industries, from agriculture to healthcare, banking, manufacturing, transportation, energy, and security. The pervasiveness of robotics technologies, from airplane autopilots, to bank machines, to smart-phones, to self-driving cars helps students to be “engaged learners” as they believe that the content that they are studying is important or will be valuable to them.

The Introduction to Programming curriculum is just that: an introduction. For many teachers this will be your first experience at teaching robotics and programming. If you need help, the Robotics Academy has lots of free resources on its website and regularly offers teacher courses. If you have questions or find issues, we would love to hear from you.

Enjoy your school year.

Robin Shoop,
Robotics Academy Director

Jesse Flot,
Lead Writer,
ROBOTC EV3 Curriculum
# Table of Contents

## Introduction

- EV3 Lesson Structure
- What does the EV3 Curriculum Teach?
- Differentiated Instruction
- General Layout of the Introduction to Programming EV3 Curriculum
- How do I use the Curriculum?
- What topics are covered in each Unit?

## Standards

- Common Core Math Practices
- Common Core Math Content
- Common Core English/Language Arts
- Next Gen Science Standards
- Computer Science Principles

## In the classroom

- How should student work stations be setup?
- What are the System Requirements?
- Scope and Sequence
- Six Week Implementation Schedule
- Nine Week Implementation Schedule
- Twelve Week Implementation Schedule
- Eighteen Week Implementation Schedule
- Physical and Virtual Programming Challenges
- Search and Rescue Challenge

## Teacher Notes

- The Getting Started Chapter
- The Basic Movement Chapter
- The Sensors Chapter
- The Program Flow Chapter

## 43 First Day Activities

- Introduction to Robot Programming
- Teaching How to Troubleshoot Programs
- Teaching Pseudocode and Flowcharts
- Introduction to Pseudocode Lesson Set
- Pseudocode Exercise
- Introduction to Flowcharts Lesson Set
- Flowchart Exercise
- The Tic-Tac-Toe Challenge

## 60 End of Chapter Programming Challenges

- Basic Movement Straight and Turning Reflection Questions
- EV3 Turning Investigation
- Physical Robot Basic Movement Challenges
- Physical Robot Touch, Distance, Gyro, and Color Sensor Programming Challenges
- Physical Robot Program Flow Challenges
- Virtual Robot Basic Movement Challenges
- Search and Rescue Challenge
- Virtual World Touch, Distance, Gyro, and Color Sensor Programming Challenge
- Virtual World Program Flow Challenges

## 88 Engineering Activities

- Project Planning: Engineering Journals
- Project Planning: PERT/Gantt Charts
- Project Planning: Design Reviews
- Robot Maze Challenge, Robot Product Design Challenge

## 97 Assessment Rubrics

- Work Habit Evaluation Rubric, Writing Rubric, Presentation Rubric, Design Review Rubric, and Proposal Writing Rubric
- Robot Physics Terms Handout
- Technical Sketching
- Engineering Process
Frequently Asked Questions

Before starting

► Will *Introduction to Programming* help me teach to Standards?
  Yes! See Standards, pages 10 - 12.

► What do I need to prepare for class?

► What general topics are covered with the curriculum?
  See Topics Covered, page 9 and the Scope and Sequence on pages 14-17.

► What is the EV3 Curriculum’s the lesson structure?
  See EV3 Lesson Structure, page 5.

► I want to know what’s in each Chapter and Unit, where to I go?
  See Getting Started pages 24-26; Basic Movement, pages 27-31; Sensors, pages 32-36; and Program Flow, pages 37-42;

During class

► How do I teach kids to be independent troubleshooters?
  See “Teaching How to Troubleshoot Programs” pages 46-47.

► What should I teach and when should I teach it?
  See the suggested Implementation Schedules for a 6, 9, 12, 15, or 18 week robotics class, pages 18-21.

► What do I do about students who go faster/slower than the others?
  See Differentiated Instruction, page 6.

After class

► How do I prepare my kids for competitions?
  See Engineering Activities 88-94.

► Are there quizzes or homework?
  Chapters include reflection questions and programming challenges designed to let students apply their skills and knowledge.
Checklist

- **Identify the Goals of your Robotics Course**
  Robotics can be used to teach to lots of standards. This curriculum is designed to introduce students to how to program, an important part of robotics, but not the only thing that you can teach through robotics. Read the Implementation Plans, pages 18-21 and think about what you want to teach and how you want to teach it.

- **Set up the student workstations**
  See page 13, Workstation Setup.

- **(Recommended) Build the Driving Base for each robot**
  Since mechanisms aren’t the focus of this module, pre-building the basic robot for your students can save multiple weeks of class time and allow them to begin work immediately on Day 1. The plans can be found in the Basic Movement Chapter, Moving Straight Unit.

- **Become familiar with the lessons**
  See page 5 to become familiar with the lesson flow. Read the Scope and Sequence, pages 14-17, to become familiar with the content flow. For a more detailed description read the chapter reviews (i.e. Basic Movement pages 27-31).

- **Determine overall pacing for the module**
  Identify key dates that you would like to have each project due by; make these clear to students in your syllabus or assignment sheets.

- **(Highly Recommended) Read “Teaching How to Troubleshoot Programs”**
  See pages 46 and 47.
Guided Robot Programming Activities, Extension Activities for Advanced Students, and End of Chapter Open-Ended Problems

The Introduction to Programming the EV3 Curriculum is a curriculum module designed to teach core computer programming logic and reasoning skills using a robotics context. The curriculum consists of three chapters (Basic Movement, Sensors, and Program Flow) and each chapter is broken into units that teach key robotics and programming concepts. Additionally, there is a huge amount of support for teachers competing in Robotics Competitions for the first time included in this teacher’s guide!

Each project comprises a self-contained instructional unit in the sequence, and provides students with:

► An introduction to a real-world robot and the context in which it operates
► An explanation of how the robot solves the problem
► A LEGO EV3 - scale version of the problem to solve with a EV3 robot
► Step-by-step guided video instruction that introduces key lesson concepts (e.g. Loops) by building simple programs that progress toward the end of unit programming challenge
► Built-in questions that give students instant feedback on whether they understood each step correctly, to aid in reflection and self-pacing
► Semi-guided “Try It!” exploration activities that expose additional uses for and variants on each robot behavior
► Semi-open-ended Mini-Challenges which ask students to use the skill they have just learned to solve a relevant small portion of the final unit challenge
► The Unit Challenge based on the original robot’s problem, for students to solve in teams as an exercise and demonstration of their mastery of the concept
► Robot Virtual World extension activities. The RVW activities are designed to significantly enhance student’s programming opportunities allowing them to program robots underwater, on an island, or in an outer space environment using the same commands that they use to program their LEGO EV3 physical robot.
Introduction

What does the EV3 Curriculum teach?

► How to control basic robot movements
  a. Robot math
  b. Sequences of commands

► Sensors and how they work
  a. Touch sensor, sonar sensor, gyro sensor, and color sensor

► Intermediate concepts of programming
  a. Program Flow Model
  b. Wait Until Commands
  c. Decision-Making Structures
     • Loops
     • If/Else
     • Repeated Decisions

► Teach troubleshooting strategies and engineering practices
  a. Problem-solving strategies
  b. Teamwork

Differentiated Instruction

One of the biggest challenges facing teachers today is meeting the needs of each individual student in their classroom; that is the core of differentiated instruction. Differentiated instruction asks teachers to approach students at their instructional level, and requires students to show evidence of growth from their instructional level. Differentiated instruction encompasses more than just assessment. It involves all aspects of instruction: classroom delivery, overall learning environment, learning content, and assessment. The EV3 programming curriculum and ROBOTC software provides many opportunities for students of all abilities:

► Different math levels - Solve the Expedition Atlantis Math Game in Explorer, Cadet, and Admiral Levels.

► Solving the open-ended programming challenges embedded into the units that make up the Movement, Sensors, and Program Flow Chapters.

► Completing the virtual programming challenges found in the Robot Virtual World games found within the LEGO EV3 curriculum (Ruins of Atlantis, Palm Island, Operation Reset): attempt to complete the entire world, or choose to program different robots within a Virtual World.

► Challenging gifted students to move from graphical to text based ROBOTC.

► Working cooperatively with students having difficulty grasping some concepts.

► Engaging in engineering challenges that are found in robotics competitions.
Introduction to Programming the EV3 with ROBOTC Graphical

Pictured at the left is the main menu. Click any section to open a Unit in a Chapter, or click a page number to go directly to the page.

**Getting Started**

**System Configuration:** Set up the robot and learn about its basic operation and maintenance

**Your First Programs:** How to get started using either physical or virtual robots

**Basic Movement**

**Expedition Atlantis:** Teaches robot math

**Movement Chapters:** Use sequential commands to make the robot move and turn

**Ruins of Atlantis:** Students solve a underwater programming challenge

**Sensors**

**Forward Until Chapters:** Teaches students how to program with touch, sonar, gyro, and color sensor

**Palm Island:** A robot programming challenge

**Program Flow**

**Program Flow Chapters:** Teaches students the logic used with loops and conditional statements

**Search and Rescue Challenge:** Reinforces using flowcharts and iterative program development

**Operation Reset:** A robot programming challenge
Introduction

Using the ROBOTC Introduction to Programming the EV3 Curriculum in class

*Introduction to Programming* is designed for student self-pacing in small groups, preferably pairs that are working together at one computer with one EV3 robot. It can also be used in “virtual mode” where students are learning programming using a virtual robot that is programmed using the exact same commands that they will use to program their actual EV3.

Curriculum tasks are designed to involve some – but not extensive – mechanical consideration, so that hands-on design tasks may remain authentic without becoming logistically difficult.

Solutions will not require parts in excess of those included in the EV3 core set, so it is sufficient to leave each team with one kit (although access to additional parts may allow students to construct more creative solutions to problems).

A typical plan for an *Introduction to Programming* chapter is:

1. View the introductory video as a class, or in individual groups, then review the challenge task for the unit
2. In a group, identify and note key capabilities the robot must develop, and problems that must be solved in individual engineering journals or class logs (e.g. on sticky paper posted on the walls)
3. Students proceed through the curricular training materials at their own pace, following the video instruction directly, and constructing solutions to the Try It! and Mini-Challenge steps as they go
4. Each group constructs its own solution to the Unit Challenge
   - Students may be asked to document their solutions in journals or logs, and especially to explain how they overcame the key problems identified at the start of the unit
5. Assign are teacher selected and based on the focus of the class
   - There are many handouts and challenges that allow teachers to extend the use of the curriculum to teach engineering practices, prepare for robotic competitions, or teach computational thinking generally
   - Students can complete reflection questions and other exercises that align with various chapters and are found in the reproducibles section of the teacher’s guide
### Topics are Covered in Each Unit

<table>
<thead>
<tr>
<th>Unit Name</th>
<th>Main Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Getting Started Chapter</td>
<td></td>
</tr>
<tr>
<td>System Configuration</td>
<td>A basic introduction to the software and hardware that you will use with this curriculum</td>
</tr>
<tr>
<td>Your First Program (Physical)</td>
<td>Teaches how to download and run robots using a physical environment</td>
</tr>
<tr>
<td>Your First Program (Virtual)</td>
<td>Teaches how to download and run robots using a virtual environment</td>
</tr>
<tr>
<td>Basic Movement Chapter</td>
<td></td>
</tr>
<tr>
<td>Expedition Atlantis</td>
<td>A Robot Math programming game</td>
</tr>
<tr>
<td>Moving Straight</td>
<td>Programming and configuring the forward and movement command blocks</td>
</tr>
<tr>
<td>Turning</td>
<td>Programming and configuring the turn and movement command blocks</td>
</tr>
<tr>
<td>Ruins of Atlantis</td>
<td>An underwater robot programming environment</td>
</tr>
<tr>
<td>Sensors Chapter</td>
<td></td>
</tr>
<tr>
<td>Forward Until Touch</td>
<td>How to configure and program the touch sensor</td>
</tr>
<tr>
<td>Forward Until Near</td>
<td>How to configure and program the sonar sensor (ultrasonic sensor)</td>
</tr>
<tr>
<td>Turn for Angle</td>
<td>How to configure and program the gyro sensor</td>
</tr>
<tr>
<td>Forward Until Color</td>
<td>How to configure and program the color sensor</td>
</tr>
<tr>
<td>Palm Island</td>
<td>A tropical island that allows students to practice everything they’ve learned</td>
</tr>
<tr>
<td>Program Flow Chapter</td>
<td></td>
</tr>
<tr>
<td>Loops</td>
<td>Students learn about loops</td>
</tr>
<tr>
<td>If/Else Structures</td>
<td>Introduces If and If/Else structures</td>
</tr>
<tr>
<td>Repeated Decisions</td>
<td>Introduces While Loops</td>
</tr>
<tr>
<td>Line Tracking</td>
<td>Teaches how to program and configure the lineTrack command block</td>
</tr>
<tr>
<td>Search and Rescue</td>
<td>A culminating programming challenge</td>
</tr>
<tr>
<td>Operation Reset programming game</td>
<td>An Outer Space world that challenges student to use logic and programming</td>
</tr>
</tbody>
</table>
# Math Standards Addressed

## Common Core Mathematics Practices

<table>
<thead>
<tr>
<th>Standard (CCSS.Math.Practice)</th>
<th>Introduction to Programming the EV3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MP1</strong> Make sense of problems and persevere in solving them</td>
<td>Chapters are all based around solving real-world robot problems; students must make sense of the problems to inform their solutions</td>
</tr>
<tr>
<td><strong>MP2</strong> Reason abstractly and quantitatively</td>
<td>Programming requires students to reason about physical quantities in the world to plan a solution, then calculate or estimate them for the robot</td>
</tr>
<tr>
<td><strong>MP4</strong> Model with mathematics</td>
<td>Many processes, including the process of programming itself, must be systematically modeled on both explicit and implicit levels</td>
</tr>
<tr>
<td><strong>MP6</strong> Attend to precision</td>
<td>Robots require precise (and accurate) input, or their output action will be correspondingly sloppy</td>
</tr>
<tr>
<td><strong>MP7</strong> Look for and make use of structure</td>
<td>Understanding the structure of the physical environment, the interrelated components of robot hardware and software, and commands within a program are vital to successful solutions</td>
</tr>
<tr>
<td><strong>MP8</strong> Look for and express regularity in repeated reasoning</td>
<td>Any programmed solution to a class of problems relies on the programmer recognizing and exploiting important patterns in the problem structure. There is also an emphasis throughout the module on recognizing common programmatic patterns, as well as patterns within a solution that invite the use of Loops.</td>
</tr>
</tbody>
</table>

## Common Core Mathematics Content

<table>
<thead>
<tr>
<th>Standard (CCSS.Math.Content)</th>
<th>Introduction to Programming the EV3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>6.RP.A.1</strong> Understand the concept of a ratio and use ratio language to describe a ratio relationship between two quantities</td>
<td>Students use ratio language to describe and make use of the relationship between quantities such as Wheel Rotations and Distance Traveled</td>
</tr>
<tr>
<td><strong>6.RP.A.2</strong> Understand the concept of a unit rate a/b associated with a ratio a:b with b≠0, and use rate language in the context of a ratio relationship</td>
<td>The relationship between Wheel Rotations and Distance Traveled is a rate, customarily understood through a unit rate such as &quot;# cm per rotation&quot;.</td>
</tr>
<tr>
<td><strong>6.R.A.3</strong> Use ratio and rate reasoning to solve real-world and mathematical problems</td>
<td>Students are required to apply ratios and rates when they build their prototype examples of their real world robots.</td>
</tr>
<tr>
<td><strong>7.RP.A.3</strong> Use proportional relationships to solve multi-step ratio and percent problems.</td>
<td>Comparisons between rate-derived quantities are common during robot navigation tasks.</td>
</tr>
</tbody>
</table>
# Language Arts and Science Standards Addressed

## Common Core English Language Arts

<table>
<thead>
<tr>
<th>Standard (CCSS.ELA-Literacy)</th>
<th>Introduction to Programming the EV3</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHST.6-8.1 Write arguments focused on discipline-specific content. [See also: WHST.6-8.1.a to WHST.6-8.1.e]</td>
<td>Reflection Questions ask students to analyze, evaluate, and synthesize arguments in response to robotics and programming problems</td>
</tr>
<tr>
<td>WHST.6-8.4 Produce clear and coherent writing in which the development, organization, and style are appropriate to task, purpose, and audience.</td>
<td>Reflection Question tasks include composing technical critiques, technical recommendations, and creative synthesis.</td>
</tr>
</tbody>
</table>

## Next Generation Science Standards (NGSS)

<table>
<thead>
<tr>
<th>Standard</th>
<th>Introduction to Programming the EV3</th>
</tr>
</thead>
<tbody>
<tr>
<td>MS-ETS1-2. Evaluate competing design solutions using a systematic process to determine how well they meet the criteria and constraints of the problem.</td>
<td>Solving challenges requires students to create and evaluate both hardware and software designs according to scenario scoring criteria. Some Reflection Questions require students to make recommendations between competing alternatives based on criteria that they define.</td>
</tr>
<tr>
<td>MS-ETS1-4. Develop a model to generate data for iterative testing and modification of a proposed object, tool, or process such that an optimal design can be achieved.</td>
<td>When solving more difficult and complex challenges, students are guided toward iterative testing and refinement processes. Students must optimize program parameters and design.</td>
</tr>
<tr>
<td>HS-ETS1-2. Design a solution to a complex real-world problem by breaking it down into smaller, more manageable problems that can be solved through engineering.</td>
<td>Problem Solving methodology for challenges directs students to break down large problems into smaller solvable ones, and build solutions up accordingly; challenges give students opportunities to practice, each of which is based on a real-world robot</td>
</tr>
<tr>
<td>HS-ETS1-3. Evaluate a solution to a complex real-world problem based on prioritized criteria and trade-offs that account for a range of constraints, including cost, safety, reliability, and aesthetics as well as possible social, cultural, and environmental impacts.</td>
<td>Some Reflection Questions require students to make recommendations about real-world policies (e.g. requiring sensors on automobiles) based on the impact of that decision</td>
</tr>
</tbody>
</table>
## Computer Science Principles Framework (CSP)

<table>
<thead>
<tr>
<th>Learning Objective</th>
<th>Introduction to Programming the EV3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1.1 Use computing tools and techniques to create artifacts. [P2]</td>
<td>Challenge activities result in the creation of a (simple) algorithmic solution and an accompanying program that implements it.</td>
</tr>
<tr>
<td>1.1.2 Collaborate in the creation of computational artifacts. [P6]</td>
<td>Students work in teams to accomplish tasks.</td>
</tr>
<tr>
<td>1.1.3 Analyze computational artifacts. [P4]</td>
<td>Students perform debugging on their own code, as well as analyze and evaluate others’ code and suggested code in Reflection Questions.</td>
</tr>
<tr>
<td>1.3.1 Use programming as a creative tool. [P2]</td>
<td>Students use programming to solve model challenges based on challenges real robots face.</td>
</tr>
<tr>
<td>2.2.1 Develop an abstraction. [P2]</td>
<td>Robots gather information about the world through sensors, which turn physical qualities of the world into digital abstractions. Students must understand and work with this data to develop then implement their solution algorithms.</td>
</tr>
<tr>
<td>2.3.1 Use models and simulations to raise and answer questions. [P3]</td>
<td>Students construct and use a “program flow” model of programming itself to understand how the robot uses data to make decisions and control the flow of its own commands.</td>
</tr>
<tr>
<td>4.1.1 Develop an algorithm designed to be implemented to run on a computer. [P2]</td>
<td>Students develop solution algorithms to each challenge and mini-challenge problem before implementing them as code. Reflection Questions also ask students to evaluate algorithms expressed as pseudocode.</td>
</tr>
<tr>
<td>4.2.1 Express an algorithm in a language. [P5]</td>
<td>Students develop code to robotics challenges in the EV3 Programming Language.</td>
</tr>
<tr>
<td>5.1.1 Explain how programs implement algorithms. [P3]</td>
<td>Students must communicate solution ideas within groups and as part of class discussion, as well as in Reflection Questions.</td>
</tr>
<tr>
<td>5.3.1 Evaluate a program for correctness. [P4]</td>
<td>Students test and debug their own code, and evaluate others’ in the Reflection Questions.</td>
</tr>
<tr>
<td>5.3.2 Develop a correct program. [P2]</td>
<td>Programmed solutions to challenges must work.</td>
</tr>
<tr>
<td>5.3.3 Collaborate to solve a problem using programming. [P6]</td>
<td>Students develop solutions in teams.</td>
</tr>
<tr>
<td>5.4.1 Employ appropriate mathematical and logical concepts in programming. [P1]</td>
<td>Relationships such as “distance per wheel rotation” are important to making solutions work.</td>
</tr>
<tr>
<td>7.4.1 Connect computing within economic, social, and cultural contexts. [P1]</td>
<td>Reflection Questions ask students to make evaluative recommendations based on the impacts of robotic solutions in context.</td>
</tr>
</tbody>
</table>
Classroom Setup

What is the best setup for student workstations?

Ideally, each pair of students will work together at one computer, with one EV3 robot.

Set up each workstation with:

a. **ROBOTC 4 for MINDSTORMS Software, The Introduction to the EV3 Programming Curriculum, and Robot Virtual World Software** installed on each computer
   - Check each computer to see that Robot Virtual World software works

b. Access to the **ROBOTC Introduction to Programming LEGO** software
   - This can be installed locally or on a local network server with proper licensing
   - Students may also log into the software from home via the Carnegie Mellon Robotics Academy server.

c. Two pairs of **headphones** with **headphone splitters**
   - One pair for each student
   - Avoid using speakers, as multiple workstations in the same classroom will generate too much overlapping noise

d. One **EV3 kit per work station**

What are the System Requirements for the Introduction to Programming the EV3 Curriculum?

**Introduction to Programming the EV3 Curriculum**

- HTML5-compatible browser (Firefox, Chrome, Internet Explorer 10+)
- Tablets (iPad, Android, Windows) with HTML5 browsers should work as well, if accessing the curriculum from the Internet

**Robot Virtual World Software**

- PC Compatible OS: with an Intel Core 2 processor family or better
- Memory: 2 GB RAM
- Graphics: NVIDIA® 8800GTS or better, ATI Radeon™ HD 3850 or better
- Hard Drive: 1.5 GB free hard drive space to install all virtual worlds
What topics are covered in each Chapter?

Chapter: Getting Started

1. System Configuration
   a. The EV3 brain, sensors, ports, how to update the Linux Operation System, how to download software, and how to navigate the LEGO EV3 menus.

2a. Lesson: Your First Program (Physical Robot)
   a. How to use ROBOTC to write your first program, choosing the correct platform, compiler target and Virtual World, how to open a new file, how to use the Graphical Functions area, how to drag blocks into the programming area, how to use the forward command block, how to save your program
   b. How to download and run a program on a physical EV3 robot.
   c. How to use the Motors and Sensors Setup window to program customized robots

2b. Your First Program (Virtual Robot)
   a. How to use ROBOTC to write your first program, choosing the correct platform, compiler target and Virtual World, how to open a new file, how to use the Graphical Functions area, how to drag blocks into the programming area, how to use the forward command block, how to save your program
   b. How to download and run a program, how to select your robot in the Robot Virtual World, how to reset a program in the Robot Virtual Worlds
   c. How to use the Camera Controls in the Virtual World, how to zoom in or out in the Virtual World
   d. How to use the Measurement Toolkit that is part of the Virtual World software

Chapter: Basic Movement

1. Lesson: Moving Forward
   a. Introduction to Basic Movement and the Sensabot challenge
   b. Robot Configuration
   c. What each box on the Forward command does, how to make the robot move a precise distance, how to make the robot move backwards, how to use timing, changing the robots speed
   d. How to raise the robot arm, how to use the moveMotor command block, what each box in the moveMotor command block does, how to lower the robot arm.
   e. Program review
   f. Challenge Overview
Scope and Sequence

2. **Lesson: Turning**
   a. Introduction to Turning and the Autonomous Tractor Challenge
   b. Robot Configuration
   c. How to program the robot to make a right turn, how to use the turnRight block, what each box in the turnRight block does, how each individual motor movement causes the robot’s body to turn
   d. Program review
   e. Challenge Overview

**Chapter: Sensors**

1. **Lesson: Forward Until Touch**
   a. How robots use sensors to respond to the environment, Introduction to the Touch Sensor.
   b. Robot Configuration
   c. How to use the setMultipleMotors block, what each box in the setMultipleMotors block does, how to use the waitUntil block, what each box in the waitUntil block does, what values a Touch Sensor returns, how to use the stopMultipleMotors block, what each box in the stopMultipleMotors block does, how to program a robot to move forward until the Touch Sensor detects an object.
   d. Big Ideas 1 & 2 - Programming is Precise, and Sensors, Programs, Actions
   e. Program review
   f. Introduction to the Arm Position Challenge

2. **Lesson: Forward Until Near**
   a. Introduction to the Hexarotor, the Ultrasonic Sensor, and the Ultrasonic Maze Challenge
   b. Robot Configuration
   c. How to program a robot to move forward until the Ultrasonic Sensor detects an object, the unit that the Ultrasonic Sensor uses when detecting an object, and how to use the value from the Ultrasonic Sensor to control the robot’s motors.
   d. Program Review
   e. Introduction to the Maze challenge

3. **Lesson: Turn for Angle**
   a. Introduction to the Autonomous Lawn Mower, the Gyro Sensor, the advantages of using the Gyro Sensor, and introduce students to the Golf Course Mower Challenge,
   b. Robot Configuration
   c. How the Gyro Sensor measures turns, how to use the resetGyro block, what each box in the resetGyro command does, how to use the setMotor blocks, what each box
in the setMotor command block does, which port the left motor is plugged into, which port the right motor is plugged into, how to program and use the Gyro Sensor to make 90 degree left turns

d. The causes of the robot “overturning”, how to compensate for “overturning”, and how to get the robot to stop closer to 90 degrees

e. Program Review

f. Introduction to the Golf Course Mower Challenge

4. Lesson: Forward Until Color

   a. Introduction to the Traffic Signal Challenge, introduction to the Color Sensor

   b. Robot Configuration

   c. How to program a robot to move forward until the Color Sensor detects the color red, introduction to the different modes of the color sensor, how to program your robot to stop when the Color Sensor detects a color.

   d. Program review

   e. Introduction to the Traffic Signal Challenge

Chapter: Program Flow

1. Lesson: Loops

   a. Introduction to the Container Transporter, Introduction to Loops

   b. Robot Configuration

   c. How to use a Repeat Loop to run the same sequence of commands multiple times, how to add a Loop block to a program, how to program a robot to move back and forth forever

   d. How to use a Loop with count control, how to modify the Repeat Loop, how to program a robot to move back and forth for a specific amount of times

   e. How to use a repeatUntil block with a sonar sensor, how to configure the repeatUntil block for this program, how the repeatUntil block and the sonar sensor work together in this program, and how to test the program

   f. Big Idea 3 - Make Sense of Systems

   g. Program review

   h. Introduction to the Container Transporter Challenge

2. Lesson: If/Else Conditional Blocks

   a. Introduction to the Strawberry Plant Sorter Robot, Introduction to If/Else Conditional Blocks and how they make decisions, and introduction to the chapter challenge

   b. Robot Configuration

   c. How to add an If/Else Conditional Block to a program, explanation of the branches in an If/Else Conditional Block, how the If/Else Command Block makes a decision, how
to program the robot to make a decision based on feedback from the Sonar Sensor
d. How to program the robot with an If/Else Conditional Block and a Repeat Forever block to control a robot’s behavior
e. Program review
f. Introduction to the Strawberry Sorter Challenge

3. **Lesson: Repeated Decisions**

a. Introduction to the Autonomous Tractor Robot, Introduction to Orchard Tractor Obstacle Challenge

b. Robot Configuration
c. Identify common logic mistakes that new programmers make with Obstacle Detection, identify why it is important to use the setMultipleMotors command block with Obstacle Detection, learn how to use encoder values with the Repeat Until Loop block, learn how programming blocks can hold up program flow preventing a sensor from being continuously checked
d. How to think of Obstacle Detection in terms of small repeated decisions, how to use a combination of the If/Else conditional block and the repeatUntil block to make repeated decisions, review how repeated decisions allow a robot to check each sensor in turn, without either one blocking the other
e. Program Review
f. Introduction to Orchard Obstacle Challenge

4. **Lesson: Line Tracking**

a. Introduction to Automated Material Transport System, Introduction to a repeated conditional behavior

b. Robot Configuration
c. How the robot should react to different Color Sensor reading while line tracking, introduction to lineTrackLeft and lineTrackRight command blocks, how to use a lineTrack command block with a Repeat block, and how to use a combination of the lineTrack and repeatUntil Blocks to track a line for a specific number of rotations
d. Program Review
e. Introduction to the Line Tracking Challenge

5. **Search and Rescue Challenge**

a. Introduction to the Search and Rescue Challenge

b. Flowcharts Video - “How do robots think?” This video introduces students to how to use flowcharts.
c. Iterative Design Video - “What is the best way to build your program?” This video provides students with a strategy to build and test their programs.
d. Big Ideas 4 & 5 - Break down problems and build up solutions and Computational Thinking applies everywhere
e. Introduction to the Search and Rescue Challenge
### Six Week Implementation Schedule

Following is a 6 week implementation schedule for Introduction to Programming the EV3. Obviously, in 6 weeks it will be impossible to teach all of the concepts included in the curriculum. The 6 week implementation schedule begins with the Getting Started chapter and goes through the Basic Movement chapter. Much more detail on each of the chapters is provided in the chapter overview section beginning on page 23 of this document. Teachers are able to stay with this schedule as it is written, or replace certain parts with other activities. The EV3 curriculum contains many different Extension Activities for further practice that can be used at your discretion.

<table>
<thead>
<tr>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Expedition Atlantis</strong></td>
<td><strong>Getting Started: About EV3 and Your First Program</strong></td>
<td><strong>Moving Forward</strong></td>
</tr>
<tr>
<td>See Expedition Atlantis Teacher’s Manual for an in depth explanation of the week-long implementation of Expedition Atlantis Robot Math Game.</td>
<td>1. Use the Getting Started Unit to introduce students to the EV3, ROBOTC Software, and Robot Virtual Worlds. 2. Help students program using the ROBOTC interface by complete the Getting Started activities. 3. Monitor students as they download and run their first program. Activities: About the EV3 Video. Programming the EV3 robots video. ROBOTC Graphical Interface video. Your first program: Making your robot move forward. Building with EV3 Handout. Introduction to programming, Introduction to Pseudocode.</td>
<td>1. Assign students to complete the Moving Forward unit. 2. Have students describe how they calculate the number of rotations a robot needs to move to travel a specific distance. 3. Complete all of the chapter activities in the Moving Forward Chapter. The Try It! Activities The Mini Challenges And the Optional Challenges found at the bottom of lessons 3 &amp; 4.</td>
</tr>
<tr>
<td><strong>Week 4</strong></td>
<td><strong>Week 5</strong></td>
<td><strong>Week 6</strong></td>
</tr>
<tr>
<td><strong>Turning</strong></td>
<td><strong>Movement and Turning</strong></td>
<td><strong>Introduction to Sensors</strong></td>
</tr>
<tr>
<td>1. Assign students to complete the Turning Chapter. 2. Have students describe the math they use to calculate robot turns. 3. Complete the Orchard Tractor Challenge. Note: you may want your students to watch the “Iterative Design” Video, Lesson 3, Search and Rescue. Activities: Turning Chapter Videos, Try It! Activities and Mini Challenges and Optional Challenges found at the bottom of lessons.</td>
<td>Week 5 is designed to allow all students to complete the Movement Unit and the Expedition Atlantis Robot Math Game. Pages 60 - 64 Moving Straight and Turning Investigations Complete the Ruins of Atlantis Robot Virtual World Game. See the Ruins of Atlantis Teacher’s Guide</td>
<td>1. Assign students to complete the Forward Until Touch Unit and all of the activities in the unit. 2. Assign students to complete the Arm Position Challenge. Activities: Try It! Already Pressed, EV3 Buttons, and Forward Until Released, Mini Challenge 1: Vacuum, The Arm Position Challenge, Optional Activities at the bottom of Lesson 3. Review and discuss the Big Ideas video. Complete the Arm Position Challenge.</td>
</tr>
</tbody>
</table>
# Implementation Schedules

## Weeks 7 - 9 on a Nine Week Implementation Schedule

The 9 week schedule builds on weeks one through six. The 9 week implementation schedule teaches students basic programming skills, robot math, and introduces students to sensors. Teachers can use the schedule as it is written, or modify the content based on their individual needs. This guide contains many different Extension Activities for further practice. For example, a teacher might want the students to have more practice after the Basic Movement chapter and thus have the students use the Ruins of Atlantis Robot Virtual World.

<table>
<thead>
<tr>
<th>Week 7</th>
<th>Week 8</th>
<th>Week 9</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Forward Until Near</strong></td>
<td><strong>Turn for Angle</strong></td>
<td><strong>Forward Until Color</strong></td>
</tr>
<tr>
<td>1. Introduce students to the Ultrasonic Sensor and discuss the advantages of using the Ultrasonic Sensor to solve the Maze Challenge</td>
<td>1. Introduce students to the Gyro Sensor, the advantages of using the Gyro Sensor, and introduce them to the end of chapter Golf Course Mower Challenge.</td>
<td>1. Introduce students to the Color Sensor, how it can be used, and the end of unit Traffic Signal Challenge.</td>
</tr>
<tr>
<td>2. Assign students to complete lesson 3 where they will learn how to program a robot to move forward until the Ultrasonic Sensor detects an object.</td>
<td>2. Assign students to complete the Turn for Angle unit. Teach students how the Gyro Sensor measures turns, how to use the resetGyro block, and what each box in the resetGyro command does.</td>
<td>2. Assign students to complete the Forward Until Color unit. Teach students: how to program a robot to move forward until the Color Sensor detects the a color, the different modes of the color sensor, and the different output values of the different modes of the Color Sensor.</td>
</tr>
<tr>
<td>3. Assign students to describe what a threshold value is - Mini Challenge 1: Threshold Values.</td>
<td>3. Assign students to write programs that solve multiple types of turns using feedback from the Gyro Sensor.</td>
<td>3. Assign students to write a program where the robot stops when the Color Sensor detects a color and then continues when the color is no longer present.</td>
</tr>
<tr>
<td>4. Assign students to complete the Maze Challenge.</td>
<td>4. Discuss the causes of the robot “overturning”, how to fix “overturning”, how to program and use the Gyro Sensor to make 90 degree right turns, and why a programmer cannot just use turnLeft or turnRight commands with the Gyro Sensor.</td>
<td>4. Change the configuration of the Color Sensor so it is pointing at the ground and assign them to write a program that stops the robot when it sees a specific color.</td>
</tr>
<tr>
<td>Activities:</td>
<td><strong>Activities:</strong></td>
<td><strong>Activities:</strong></td>
</tr>
<tr>
<td>Mini Challenge 1: Threshold Values</td>
<td>Mini Challenge 1: Threshold Values</td>
<td>Mini Challenge 1: Right Turns</td>
</tr>
<tr>
<td>Mini Challenge 2: Move Until Far</td>
<td>Mini Challenge 2: Gyro Lap</td>
<td>Mini Challenge 2: Gyro Lap</td>
</tr>
<tr>
<td>Complete the Dynamic Maze Challenge</td>
<td>The Golf Course Mower Challenge</td>
<td>The Golf Course Mower Challenge</td>
</tr>
<tr>
<td>Assign the students to describe how a Ultrasonic Sensor Works</td>
<td></td>
<td>Try It! Color Sensor Values</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mini Challenge 1 Red Light, Green Light</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mini Challenge 2: Forward to Stop Line,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The Traffic Signal Challenge</td>
</tr>
</tbody>
</table>
Implementation Schedules

Weeks 10 - 12 on a Twelve Week Implementation Schedule

The twelve week implementation builds on weeks one through nine. This plan takes students from Basic Movement through Sensors and the majority of Program Flow. Teachers are able to stay with this schedule as it is written, or replace certain parts with other activities. Teachers that are using the twelve week plan are typically teaching a robotics engineering course and may be involved in a robotics competition. The focus of the description as written foregrounds programming, but if you are teaching engineering or entering a team in a robotics competition you may want to deviate from this plan and integrate some of the engineering competencies found in weeks 13 through 15 earlier. It will be up to the teacher to determine what the best plan is for their course.

<table>
<thead>
<tr>
<th>Week 10</th>
<th>Week 11</th>
<th>Week 12</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Loops</strong></td>
<td><strong>If/Else</strong></td>
<td><strong>Repeated Decisions</strong></td>
</tr>
</tbody>
</table>
| 1. Introduce students to the concept of program flow. The chapter begins by teaching students about loops and then introduces them to the Container Handling Programming Challenge. | 1. Teach students about If/Else Conditional Blocks then introduce them to the Strawberry Plant Sorter Challenge  
*note: this is a good time to review Flowcharts and Pseudocode.*  
Lesson 2 of the Search and Rescue Challenge. | 1. Introduce students to the Autonomous Tractor Robot Challenge and have them consider how they might solve the challenge. |
| 2. Teach students to use a Repeat Loop to run the same sequence of commands multiple times, how to add a Loop block to a program, and how to program a robot to move back and forth forever. (Lesson 3 in loops) | 2. Teach students how to add an If/Else Conditional Block to a program, explanation of the branches in an If/Else Conditional Block, how the If/Else Command Block makes a decision, how to program a robot to make a decision if there is a robot in front of it or not based on feedback from the sonar sensor. (Lesson 3 in If/Else) | 2. Introduce students to how to think about obstacle detection. Lesson 3 teaches an important misconception that students will need to understand to successfully solve the problem. |
| 3. Teach students how to use a Loop with count control, how to add conditions to a Loop, and how to program a robot to move back and forth five times. (Lesson 4 in loops) | 3. Teach students how to use a Repeat block with an If/Else Conditional Block, how to program a robot to solve a Maze Challenge, and how to use the EV3 LED Sensor with the If/Else Conditional Block | 3. Teach students to think about making very small decisions using the If/Else conditional block and a loop. (Lesson 4 in Repeated Decisions) |
| 4. Teach students how to use a Loop with sensor control. (Lesson 5 in Loops) | | Activities: |
| Activities: | Activities: | |
| Mini Challenge 1: Square Dance, Try It! Changing Loop Count, | Flow Charts, Mini Challenge 1: Color Sensor Comparison, | |
| Mini Challenge 2: Square Dance 2, Try It! Repeat Until Touch Sensor, | Try It! Smart Maze Solver, | |
# Implementation Schedules

## Weeks 13 - 15 on an Fifteen Week Implementation Schedule

The fifteen week implementation builds on weeks one through twelve. This plan takes students from Basic Movement through Sensors and all of Program Flow. Teachers are able to stay with this schedule as it is written, or replace certain parts with other activities. Teachers that are using the fifteen week plan are typically teaching a robotics engineering course and may be involved in a robotics competition. The focus of the description as written foregrounds programming, but if you are teaching engineering or entering a team in a robotics competition you may want to deviate from this plan and integrate some of the engineering competencies found in weeks 15-through 18 earlier. It will be up to the teacher to determine what the best plan is for their course.

<table>
<thead>
<tr>
<th>Week 13</th>
<th>Week 14</th>
<th>Weeks 15 - 18</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Line Tracking</strong></td>
<td><strong>Program Flow Challenges</strong></td>
<td><strong>Engineering Process</strong></td>
</tr>
<tr>
<td>1. Introduce students to the ATMS Robot and the Line Tracking Challenge.</td>
<td>1. Students will complete the Search and Rescue Programming Challenge.</td>
<td>Note: The focus of this curriculum is teaching programming, but the curriculum designers understand that many teachers are using robotics to teach engineering and have included the following handouts and activities in the teacher’s guide:</td>
</tr>
<tr>
<td>2. Teach students how to use and configure the lineTracking block, how to use grayscale mode, how to calculate thresholds, how to optimize your code when you are tracking a line, and basic troubleshooting.</td>
<td>2. Review all sensors and programming logic.</td>
<td>• EV3 Turning Investigation pages 63-64</td>
</tr>
<tr>
<td>Activities:</td>
<td>3. Review pseudocode and flowcharts.</td>
<td>• Project Planning: Engineering Journals pages 88-91</td>
</tr>
<tr>
<td>Try It! Right-edge Line Tracking, Try It! Line Tracking Threshold, Try It Turning Ratio, Mini Challenge 1: Line Track 4 Rotations</td>
<td>Robots to the Rescue Operation Reset Programming Game</td>
<td>• Project Planning: PERT Charts page 92</td>
</tr>
<tr>
<td>The Line Tracking Challenge</td>
<td>Operation Reset is a complex game with multiple levels and challenges. It will be up to the teacher to determine how many levels the students complete.</td>
<td>• Project Planning: Gantt Charts page 93</td>
</tr>
<tr>
<td><strong>The Search and Rescue Challenge</strong></td>
<td></td>
<td>• Project Planning: Design Reviews 94</td>
</tr>
<tr>
<td>1. Introduction to Search and Rescue Robots and the Search and Rescue programming challenge.</td>
<td></td>
<td>• Technical Sketching pages 103-104</td>
</tr>
<tr>
<td>2. How robots think. It will be important to review flowcharts now. The Search and Rescue challenge can be challenging to new programmers.</td>
<td></td>
<td>• Product Prototype Design Challenge page 96</td>
</tr>
<tr>
<td>3. How to build complex programs, how to test and iteratively improve the your programs.</td>
<td></td>
<td>• Maze Solver Challenge page 95</td>
</tr>
<tr>
<td>4. Begin the Search and Rescue programming challenge</td>
<td></td>
<td>• Robot Automation Challenge page 97</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Engineering Process Handouts pages 105-106</td>
</tr>
</tbody>
</table>
Physical and Virtual Programming Challenges

The EV3 Programming Curriculum contains both physical and virtual robot programming challenges. The challenges are designed to align with the lessons learned in the associated unit. For example, the challenges shown on this page are from the “Turning Unit”. By the time that students complete the Turning Unit they should be able to program their robot to accurately move straight and turn specific angles. In this challenge they are asked to write a program that will autonomously navigate through the orchard.

Pictured at the left is a PDF handout that explains the robot programming challenge to students solving the problem using physical robots.

Pictured above is a screen shot of the virtual challenges found in the Robot Virtual World Movement Unit.

Pictured above is the simulation environment for the virtual Orchard Challenge. (similar to the physical robot challenge above left.)

Pictured above is the Movement Unit with a different challenge selected.

Pictured above is the stylized version of the virtual Orchard Challenge. (we believe that students will find this version more engaging.)
The Search and Rescue Challenge

The culminating programming challenge for students in the Program Flow Chapter is called the Search and Rescue Challenge. Students will need to write a program for their robot that enables the robot to make a decision based on what it encounters when it enters various rooms of a building (the robot is dropped in a different spot each time the program starts. One of the four rooms contains a person to be rescued. The other three rooms contain a fire, rubble, and a clear path (unobstructed). The robot must be programmed to navigate through all 4 rooms by sensing the condition of each room and taking the appropriate actions to proceed and ultimately return to the starting point.

If the challenge is played using the Robot Virtual World software the rooms are automatically switched each time a student enters the building. This requires students to develop an algorithmic solution that requires decision making. If the challenge uses physical robots the teacher should switch the room configuration each time the student starts the challenge.

Pictured above is the PDF handout provided for students that are solving the challenge using physical robots. The dynamic challenge remains, each time the robot comes to the table the course should be switched. You many consider using dice to determine the location of the rooms.
Getting Started

Chapter Overview: Getting Started

Within the Getting Started chapter, students will learn the steps they need to go through to make the software talk to the robot. Specifically, the Getting Started Chapter discusses the following:

• That the EV3 the brain and how motors and sensors plug into the brain
• How to update the EV3 Firmware
• What Firmware is and how to download ROBOTC firmware to the EV3
• How to write code using ROBOTC’s graphical interface
• How to download the code to a EV3 physical or virtual robot
• How to run the code on the robot using a tethered connection
• How to run the code on the robot without a tethered connection
• How to use the motors and sensors setup window
• How to use camera controls in the virtual environment
• How to use the measurement toolkit in the virtual environment.

The Getting Started Chapter is divided into three units: System Configuration, Your First Program (Physical Robot), and Your First Program (Virtual Robot). Each of those units is further divided into separate lessons which teach the major concepts of the chapter.

System Configuration

The System Configuration Unit is divided into four lessons:

Lesson 1: System Configuration 1: Ports
This lesson includes a video that introduces the EV3 Brain and how the inputs and outputs are named. Following the video, students asked to identify parts of the EV3’s brain.

Lesson 2: System Configuration 2: Updating the EV3 Linux Kernel
This video shows students how to update the EV3 Linux Kernel, how to connect the robot to the computer, how to turn the robot on, how to download ROBOTC, how to set the platform type, and how to download the software.

Lesson 3: System Configuration 3: Downloading Firmware
This video explains the job of firmware as well as how to download the firmware to the EV3. At the end of the lesson students are asked the following Check Your Understanding (CYU) questions:

• What is Firmware?
• What software do you use to download firmware onto the robot?
• Where in the software is the option to download the firmware?

Lesson 4: System Configuration 4: Menus
This video explains how to access and navigate the EV3 menu system. This video shows how to turn the EV3 on and off, as well as how to run programs and see port values.
Your First Program (Physical Robots)

Lesson 1: Your First Program (Physical Robot)
1: Writing Your First Program
This lesson teaches students how to open up and use the ROBOTC Graphical programming language with a physical robot. They learn how commands are configured, how to drag commands into the programming screen, how commands are run, and how to save them using ROBOTC. At the end of the lesson students will be able to answer:

• What is the programming software
• How to properly configure the EV3
• How to build a graphical program
• How robot programs run
• How to configure the command blocks

Lesson 2: Your First Program (Physical Robot)
2: Downloading and Running
This lesson teaches students how to download and run the program on a physical robot. Students learn how to set the compiler target, how to connect the USB to the robot, and how to run the robot. At the end of the lesson students are asked the following CYU questions:

• How do you turn on the EV3?
• How do you navigate the menus on the EV3?
• How do you run a program in tethered mode?

Lesson 3: Your First Program (Physical Robot)
3: Motors and Sensors Setup
This lesson teaches students how they use the Motors and Sensors Setup configuration wizard to setup their robot. Students will learn how to change the Motors and Sensors Setup for the configuration of their robot. This lesson also includes several Try It! student extension activities:

Try It!: Removing Configuration - students are challenged to open up the Motors and Sensors Setup and to configure a different sensor in non-default port.

Try It!: Changing Default Sensor Ports - students are challenged to open up the Motors and Sensors Setup and configure their sensors to use a non-default sensor port.

Your First Program (Virtual Robots)

Lesson 1: Writing Your First Program
This lesson teaches students how to open up and use the ROBOTC Graphical programming language with a virtual robot. More on the next page.

Lesson 2: Downloading and Running
This lesson teaches students how to download and run the program on a Virtual EV3 robot. Students learn how to download a program to a virtual robot, how to log into the virtual world, how to select a challenge level, and how to run the program.

Lesson 3: Camera Controls in the Virtual Worlds
In this lesson students learn how to use the camera controls in the virtual worlds.

Lesson 4: the Measurement Toolkit
In this lesson students learn to use the measurement toolkit embedded into the virtual worlds so that they can mathematize their programming solutions.

Saving Progress and Badges

We’ve built automated assessment tools into the Robot Virtual World software. Students learn the correct way to access the virtual worlds in Lesson’s 1 and 2 in the Your First Program (Virtual Robot) Chapter (above right).

Each of the robotic challenges are badged using the Robot Virtual World software. The badges are stored in cache on each local machine and if students log in correctly they will be able to print a PDF of their accomplishments for your record.

More on the next page...
Getting Started

Saving Progress and Badges

At the left are pictures that show how students can collect badges via the Robot Virtual World software.

1. Write the program
2. Select the virtual world. In this case the world is from the Basic Movement Chapter and we are selecting the Moving Forward challenge
3. Run the challenge
4. When the challenge is completed a badge appears on the screen
5. Open up the badges screen on the RVW interface
6. Print the badge in a PDF format (below)

Pictured directly above are the badges that students earn as they work through the EV3 programming curriculum.
Basic Movement

The Expedition Atlantis Math Game

We’ve learned that kids guess and check their way through robotics activities. We’ve also learned that students struggle to learn robot math and robot programming at the same time. The Expedition Atlantis math game is designed to teach kids the mathematics behind robot movement. It requires players to consider the mathematical relationships between the robot’s wheels and its turning and straight movements.

The game is designed to place players in the role of explorers on a quest to the Lost City of Atlantis. On the way to the exploration site they run into a storm and the mission needs to be carried out by a robot that is controlled mathematically. Players (students) are required to complete one level of the Minoan of Megaliths, the Pillars of Hercules, and Poseidon’s Courtyard to fulfill the requirements of the course. The game is accompanied by a comprehensive teacher’s guide designed to generalize a student’s mathematical understanding. You will find a copy of the teacher’s guide in the appendix of this document.

Chapter Overview: Basic Movement

Within the Basic Movements chapter, students will learn the fundamentals of their robot’s movement. Specifically, the Basic Movement chapter discusses the following:

- How to make the robot move
- How to control the distance that the robot moves
- How to control the arm and claw motors
- How to make the robot turn
- How to control the robots speed.

The Basic Movement chapter is divided into two sections: Moving Forward and Turning. Each of those sections is further divided into separate lessons which illustrate the major concepts of the chapter.

Moving Forward

The Moving Forward Unit is divided into five lessons with a culminating challenge.

Lesson 1: Introduction with Sensabot This lesson includes a video that introduces the Sensabot robot. Following the video, students answer three Check Your Understanding (CYU) questions that ask students to:

- Identify why inspections of industrial facilities are important?
Basic Movement  Teacher Notes page 2 of 5

- What is the benefit of using Sensabot over human inspectors?
- What basic skills are required to complete this challenge?

CYU questions have been embedded throughout the curriculum to help students to identify key pieces of information provided within each lesson.

Lesson 2: Robot Configuration Provides students with the robot configuration necessary for the activities within this section. In this lesson the robot’s chassis will need to be equipped with the arm attachment.

Lesson 3: Moving Forward Teaches students how to write code that enables the robot to move forward. It includes a video that demonstrates how to create a program to have the robot move three rotations forward at a half speed. Students can follow along and pause as needed (or replay the video).

The Lesson 3 video is followed by three CYU questions that ask students:
- How do you add a command to the program?
- What do the first two boxes on the Forward command block control?
- And, what does the final box on the Forward command block control?

After the CYU questions, the class can convene for a planned lesson on pseudocode which will introduce the practice of listing the steps of the program in more common language before entering it into ROBOTC.

Throughout the curriculum you will find Mini-Challenges.

Mini Challenge: 85cm Distance Challenge challenges students to write a program that controls their robot to travel exactly 85cm. If your students completed the Expedition Atlantis game this gives them practice applying the math that they learned in the game. If they haven’t played the game you will need to teach them how to calculate how many rotations the robot’s wheels will need to turn for it to travel 85cm.

Another activity type integrated throughout the curriculum is a “Try It” activity. Try It activities are very short lessons that enable students to quickly modify their code to experiment with another robot behavior. Students will encounter Try Its throughout the curriculum. This lesson set includes three activities:

Try It!: Backward The student will need to change the Forward Block to a Backward Block.

Try It!: Timing-based Movement The student will change the configuration of the Block from rotations to timing.

Try it!: Speed Control The student will change the configuration of the Block to change the robot’s speed.

Lesson 4: Arm Control Teaches the student how to program the robot’s arm. The included video demonstrates how to raise the robot’s arm. Following the video, students encounter three CYU questions asking them:
- What does the moveMotor command block do?
- What happens when you use negative speed values?
- What happens when you place more than one command in a program.

Try It!: LED Feedback activity The EV3’s backlit front panel buttons provide students with visual feedback regarding when the robot has completed or is about to complete a behavior. This is an important Try It that every student needs to understand.

Another activity type is a “Did You Know” activity. They are designed to extend students learning and provide teachers with opportunities to delve into deeper STEM concepts that are not the focus of a robot programming curriculum.

The end of the Lesson 4 includes several optional Mini Challenges:

Mini Challenge 1: Cargo Retrieval The
student is challenged to write a program to move forward 50cm, drop the arm, and bring the "cargo" back to the starting point. **Did you notice? Getting the Program Stuck** The student is challenged to start the program for the Cargo Retrieval challenge with the arm up and then answer several questions.

Mini Challenges are not always optional, but in this lesson they are because they are not key components of the solution to this section’s culminating challenge. Instead, these Mini Challenges ask students to start building programs to pick up and carry objects, which are skills that will be used in many programming activities in the remainder of the unit. They are also a good review of what has been covered in the chapter up until now. Again, if some students are moving ahead, these optional activities are good opportunities to keep them engaged.

**Lesson 5: Review** provides all students with the opportunity to review their understanding of the programs written to solve the challenges in the Moving Forward unit.

**The Sensabot Challenge**

Each chapter includes a culminating challenge. The Sensabot Challenge begins with a video that explains the challenge. Below the video is a depiction of the challenge and a PDF designed to further explain the challenge.

Again, throughout this curriculum the end of unit challenges are modeled after the real robots introduced at the beginning of each unit. Each time a challenge is introduced at the end of a section, a video is included that contextualizes and explains the challenge. Below those videos are depictions of the Challenge Overviews and PDF designed to further explain the challenge. During these challenges, students have a chance to show off what they have learned!

**Turning**

Turning is divided into 4 lessons with a culminating challenge.

**Lesson 1: Introduction to the Orchard Tractor**

This lesson introduces the Orchard Tractor Challenge and includes a video that introduces students to a real world robotics application - the Autonomous Tractor. After the video, students are asked in two CYU questions asking:

- Why is it important for a robot to drive through an orchard?
- And, what is the advantage of using an Orchard Tractor over human inspectors?

**Lesson 2: Robot Configuration** explains that only the robot’s driving base is required for this section. However, the arm section equipped in the previous section can be left on the robot.

**Lesson 3: Turning in Place** Teaches students how to program the robot to turn in place. The video demonstrates how to include a turnRight command of two rotations at fifty-percent power within a program. Then it highlights the fact that the two rotations do not rotate the entire robot, but are instead two rotations of the motor/wheels. This video gives students the opportunity to add turning right to their lists of programmable commands. The video is followed by two CYU questions that ask:

- What does the robot do when the turnRight program is run?
- And, Is it true that the whole robot rotates the programmed number of times when executing a turnRight command?

**Try It! 1: Direction of Turn** Asks students to program their robot to make a left turn instead of a right one. This Try It! is followed by two CYU questions that ask students:

- How much did the robot’s wheel turn when executing this command?
- And, what does the one rotations refers to
within the turnLeft control block? These questions ensure that students now think of rotations as referring to the robot’s wheels/motors and not to the robot itself.

Try It! 2: Units of Turn Prompts students to change the second box of the command from rotations as the unit of turn to degrees as the unit of turn, and then set the number of degrees to 270. Two CYU questions follow this activity:

- How much did the robot’s wheel turn during this movement?
- And, when would degrees be a more appropriate unit than rotations.

Mini Challenge 1: 90 Degree Turn This challenge requires students to program their robot to turn an exact 90 Degree Turn.

Mini Challenge 2: Dizzy Drill A simple programming exercise.

Did You Notice? Wheels and Turning The last couple of lessons have been designed to address a common student misconception. Again, this lesson explains how the robot’s motors move when the robot is programmed to move forward, backward, turns, or spins. This lesson provides the opportunity for the teacher to talk about a common student misconception. When programming the turnRight and turnLeft command blocks using degrees as the control, many students will think that placing 90 in the box for the amount of degrees will result in a 90 degree turn. The misconception perpetuates itself when students are programming the robot in rotations or in degrees when turning.

Did You Notice? Wheel Pointers Did You Notice? Wheels and Turning These two sections are designed to further enable a student to think critically about wheel rotations and how the robot turns.

Lesson 4: Review provides an opportunity for the teacher to review how the class solved the 90 degree turn challenge.

The Orchard Tractor Challenge

This lesson begins with a video that introduces the Orchard Tractor Challenge and includes a Challenge Overview PDF designed to help students to define the problem. This lesson challenges students to program their robots to move from their starting positions through three rows of fruit trees (blocks, walls, etc.)!

A quiz is recommended to assess students’ understanding of the Turning section. A test assessing understandings of the entire Basic Movement chapter is also recommended.

The Ruins of Atlantis Game

This is not a required activity, but does give students added practice using mathematics to solve robot movement problems. The Ruins of Atlantis is a sequel to Expedition Atlantis and challenges players to program their robots and collect gems and coins in a newly discovered section of Atlantis. Players are provided with three different robots and a treasure map. The goal of the game is to collect five gems as they navigate through Atlantis.

Players will quickly recognize the need to apply mathematical solutions to their robot programming which will allow them to find and collect the maximum number of gems and coins more quickly. Players (students) are required to collect five gems using at least one robot type. Each robot has a different size chassis and different sized wheels. The game not only helps students to practice their proportional reasoning skills but also provides them with practical robotics experiences. For example, students learn how a chassis
and wheels of different sizes each impact the programming of varying robots. The guide to completing the game can be found on its page in the curriculum.

The Ruins of Atlantis Robot Virtual World programming game is not a required activity, but if you choose to use it with your class you will find that it comes with a printable user’s guide (top right), a map that allows students to calculate their way through the programming challenge (bottom left), and the game stores students game state and enables them to earn badges when they log in properly (bottom right). The programming games documentation is included in the Ruins of Atlantis User’s Guide.
Introduction to Sensors

In the Basic Movement Chapter students learned to control their robots using basic movements: forward, backward, and turning. In the Sensors Chapter, students learn to use feedback from sensors to control these movements more precisely.

Chapter Overview: Sensors

Each Sensor Unit covers a different sensor. Students will learn how each sensor works as well as how to use each sensor's feedback to control their robot's movement. Directly below is the breakdown of what is taught in each chapter:

- How to use the Touch Sensor
- How to use the Ultrasonic Sensor
- How to use the Gyro Sensor
- How to use the Color Sensor
- Palm Island Programming Challenge

Each of the first four units is subdivided into smaller activities designed to teach students about the sensor. The Palm Island Programming Challenge is an optional challenge the provides students with additional programming opportunities.

Forward Until Touch

This unit is divided into 4 lessons, a Big Idea video, and the section challenge.

Lesson 1 This lesson uses an autonomous vacuum robot to provide a real world example of uses for a touch sensor. Two CYU questions follow the video and ask students:

- Why sensors are important to robots?
- What the importance is of sensor control over basic motor commands?

Lesson 2 Provides students with plans for how their robot should be configured. If their robot is not configured this way, they will need to modify their robot to continue through this lesson set.

Lesson 3 This lesson teaches the student about the "waitUntil" command. The accompanying video demonstrates how to program the robot to move until the touch sensor is pressed and then stop both motors. Three CYU questions follow the video and ask students:

- What does the setMultipleMotors command block do?
- What does the stopMultipleMotors command block do?
- And, what does the waitUntil command block do?

Note: There is a Classroom Openers Power Point that can be used to check student's understanding of the concepts taught in the Introduction to Sensors Chapter.
Try it! 1: Already Pressed asks students to investigate what the robot will do if the touch switch is already pressed when the program begins. Encourage students to answer the question before the run the code.

Try it! 2: EV3 Buttons asks students to investigate how the buttons on the EV3’s front panel can be used as touch sensors.

Try it! 3: Forward Until Release asks students to investigate what happens when they place a box against the Touch Sensor and set the compare-to value of the waitUntil command to 0.

Mini Challenge: Vacuum asks students to program their robots to touch all four walls of a room by using its touch sensor to know when it has touched each one.

Did You Know? How the Touch Sensor Works Animation shows how sensor works.

Big Ideas 1 and 2

Between Lessons 3 and 4, students are presented with the first of the Big Idea videos. This lesson teaches: Programming is Precise and how robots use Sensors and Programming to produce actions. Students are asked five CYU questions follow the video. They are asked:

- What is meant by the statement “programming is precise”?
- What do sensors do for a robot?
- What does a robot’s program do?
- What kind of actions is a robot capable of?
- And, how does a hexarotor maintains its position?

Lesson 4 This program review enables students to check their understanding of code that can be used to solve the Vacuum Mini Challenge. Some students may ask if there are more efficient ways to write code, since this code repeats blocks of code multiple times. They will soon learn more efficient ways to write code in the Program Flow chapter.

The Arm Position Challenge - This programming challenge gives students a practical problem to that can be solved with a touch sensor. Students now have the opportunity to show off what they have learned by programming their robot to raise its arm to automatically stop (indicated by the pressing of the touch sensor by the arm), move forward for 5 rotations, grab a container, and then bring it back to the starting location. A quiz is recommended to assess students’ understanding of the Forward Until Touch section.

Forward Until Near

The Forward Until Near Unit is divided into 4 Lessons plus a culminating programming challenge.

Lesson 1 Students are introduced to the Hexarotor and learn how it navigates a maze using its sensors. Students are then asked the following CYU question:

- Why is the ultrasonic sensor preferable to the touch sensors for detecting walls and obstacles with the hexarotor.

Lesson 2 Provides students with the robot configuration of the ultrasonic sensor for this unit.

Lesson 3 Teachers students how to use and configure the waitUntil programming block to program a robot to stop when it gets close to an object. The included video demonstrates how to program the robot to move both motors at fifty percent power until the distance sensor detects an object to be less 50 cm. A CYU question follows and asks students:

- What will the robot do if the program is run while the robot is facing a wall.
Mini Challenge 1: Threshold Values This lesson challenges students to write a program that uses feedback from the Ultrasonic sensor to control the color of the EV3 LED sensor.

Mini Challenge 2: Move Until Far This lesson reinforces the concept of “what a threshold value is” and challenges the student to write code that causes the robot to stop when it is 30cm away from a wall.

Did You Know How the Distance Sensor Works This lesson explains to students how ultrasonic sensors work.

Lesson 4 Program Review This page provides an opportunity for students to check their understanding of how programs in this unit work.

The Maze Challenge This programming challenge requires students to program their robots to move from a starting area through a maze using feedback from the ultrasonic sensor to avoid touching any walls and ultimately reach the goal zone regardless of what the distances were between the walls. A quiz is recommended to assess students’ understanding of the Forward Until Near section.

Turn for Angle

The Turn for Angle Unit is divided into 5 lessons and a culminating programming challenge.

Lesson 1 This lesson introduces the autonomous golf course mower. It also introduces students to the Gyro Sensor. At the end of the video students are asked:
- What does a gyro sensor help the robot to do?
- Why does the autonomous mower use a Gyro Sensor if it already has GPS?
- And, why should your robot use a gyro sensor if it already has rotation sensors?

Lesson 2 Presents the robot configuration for this section: The driving base with the gyro sensor attached in the correct position.

Lesson 3 Teaches students how to use feedback from the gyro sensor to stop the robot’s motors when the robot turns to a specific angle. The included video shows students how to reset the Gyro sensor, then run motors opposite to one another (to create rotation), and then to configure the waitUntil block to wait until the gyro sensor detects a value greater than ninety degrees. Two CYU questions ask students:
- What does the gyro sensor measure?
- And, what actually happened when the robot ran the program?

Lesson 4 Teaches students how to compensate for the robot overturning as it did in the previous lesson. The included video explains why the robot overturns and demonstrates how to correct the turn using programming. Two CYU questions ask:
- What factors contribute to overturning?
- And, which of the work-arounds help to correct the overturning problem?

Try it!: Left Turns? Challenges students to investigate whether the same waitUntil command block works for left turns.

Mini Challenge 1: Square Box! Challenges students to program the robot to travel a full lap around a box using the gyro sensor to control all of its turns (to improve accuracy) and to include in the program commands that turn the EV3 LED orange while the robot is moving forward and green while the robot is turning.

Did You Know?: How the Gyro Sensor Works explains how the Gyros Sensor uses a Micro-Electro Mechanical System (MEMS) to determine the sensor’s heading.

Lesson 5 Provides an opportunity for students to review working solutions for the Mini Challenge: Gyro Lap.

The Golf Course Mower Challenge This challenge asks students to program their robots to drive over the entire game board. The robot can freely maneuver in straight lines, but must rely on the gyro sensor
to turn. A quiz is recommended to assess students’ understanding of the Turn for Angle section.

**Forward Until Color**

The Forward Until Color Unit is divided into 4 lessons and a culminating programming challenge.

**Lesson 1** Introduces students to autonomous motor vehicles. The included video explains what self-driving cars need to be able to accomplish in order to be functional. For example, a self-driving robotic car needs to be able to stop at traffic lights, monitor its speed, and merge into traffic. The two CYU questions that follow the video ask:

- What are some of the challenges a self-driving car needs to be able to overcome?
- And, what will your EV3 color sensor need to detect?

**Lesson 2** provides students with the robot configuration that they will use for this unit; the drive base with an ultrasonic sensor attached.

**Lesson 3** Teaches students how to program the color sensor using the waitUntil block. The video teaches students how to configure the waitUntil block to use feedback from the color sensor. Two CYU questions follow the video and ask:

- What does the program do when run?
- And, what would happen if the cube was replaced with a green cube and the same program was run?

**Mini Challenge 1: Forward to Stop Line**

Challenges students to adjust the color sensor to face downward and program the robot to use the color sensor to stop at a line on the table (to simulate stopping at a stop line on a road). This Mini Challenge is considered optional because the culminating challenge within this section does not require a downward facing sensor and is more similar to Mini Challenge 1: Red Light, Green Light, Go!. Also included is a How Does it Work?: Color Sensor optional activity.

**Lesson 4** This lesson enables students to review working solutions to the following Mini Challenges: Red Light, Green Light, Go! and Forward to Stop Line.

**The Traffic Signal Challenge**

This activity challenges students to program their robots to navigate through three different intersections, each of which has a traffic signal, the signal starts as red before switching to green. The robot does not need to stop on its own after passing through all three intersections and can instead be stopped by hand. A quiz is recommended to assess students’ understanding of the Forward Until Color section. A test assessing understandings of the entire Sensors chapter is also recommended.

**Palm Island: Lua Edition**

This game is not a required activity, but does give students added practice programming robots using sensors and mathematics. Players are given several programmable robots, a map of the island, and tasks to complete. All tasks can be completed using the programming skills that they have mastered to date. Much more information can be found about the game by opening the user guide found on the Palm Island page.
Sensors

Pictured at the left and below are some of the lesson supports found in the Palm Island Robot Virtual World challenge: A User’s Guide, a map and legend, and pictures of the digital badges the students earn as they complete the challenge.

Directly above is a picture of a student’s digital trophy case of their completed challenges. If they log into a machine and store their progress locally they can print a PDF version for the teacher to check when they have completed all assigned activities.

Pictured left is the Palm Island Robot Virtual World programming environment. Pictured right are the robot types used in Palm Island. Students receive sensor locations, the wheel diameters, and the length and width of the robots enabling them to help them to determine which robot is right for the job.
Introduction to Program Flow

Now that students can program their robots for basic movements and can use sensors to control the robot’s behavior to react to data in the robot’s environment, students are ready to learn about loops and conditional statements - which is the focus of the Program Flow chapter.

Chapter Overview: Program Flow

Within the Program Flow Chapter, students will learn how to program their robots to complete increasingly complex behaviors building on the sensor programs that they developed in the Sensor’s Chapter. The Program Flow Chapter covers the following topics:

- What is a loop?
- How loops are used to control repetitive behaviors
- How to control how many times a loop repeats itself
- How to control a loop using feedback from sensors
- How to use If-Else logic to enable robot to make decisions

- How to use the backlight on the front panel to debug code
- How to combine loops with conditional programming to create complex behaviors
- How to detect obstacles
- How to write programs that allow for continuous processing and control
- How to program the robot to follow a line by tracking with thresholds
- How to line track for distance
- How to transport by tracking

The Program Flow Chapter is divided into four units: Loops, If-Else, Repeated Decisions, and Line Tracking and a culminating Search and Rescue Challenge. Each unit is then divided into separate lessons which illustrate the major concepts of the chapter.

Loops

The Loops Unit is divided into 6 lessons, a Big Idea video, and the section challenge.

Lesson 1 Students are introduced to a container handling robot. The included
The Virtual NXT is not a LEGO® MINDSTORMS® product. LEGO Education or the LEGO Group does not sponsor, endorse, or support this product.

The video explains how the robot uses sensors and repeated behaviors to complete its repetitive function of picking up and moving large numbers of containers. The video then explains how the behaviors of this robot connects to the content of this section. The lesson then includes a CYU question that asks students:

- How can repetitive tasks be handled programmatically?

**Lesson 2** Provides students with the robot configuration for this section: the driving base with the ultrasonic sensor facing forward.

**Lesson 3** Teaches students how to program a repetitive behavior using a loop. Two CYU questions follow the video:

- What does a Loop do?
- And, How do you add a Loop to a program?

**Mini Challenge: Square Dance 1** Challenges student to use a loop to control a robot’s movement around a box.

**Lesson 4** Teaches students how to configure a Loop command enabling the loop to run for a limited number of times. Three CYU questions follow the video:

- How often will a Repeat Loop block execute its behaviors?
- What does it mean for a loop to be “Conditional”?
- And, what is the “condition” in a Loop based on?

**Try it!: Changing Loop Count** Provides an opportunity for students to modify the repeat block.

**Mini Challenge: Square Dance 2** Students are challenged to build a program using loops where the robot only travels one lap around a cube.

**Lesson 5** Teaches students how to use sensor feedback to control a repeatUntil Loop Block. They build on their understanding of “program flow”. They are also taught how to setup a threshold value. Three CYU questions follow the video:

- What condition will allow the program flow to pass?
- What is the correct type of repeat loop if you want the block to be controlled by sensor feedback?
- When does a repeatUntil block check the value of a sensor?

**Try it!: Repeat Until Touch Sensor** Provides an opportunity for students to investigate what happens when they adjust their loop programs to use the touch sensor instead of the distance sensor.

**Mini Challenge: Square Dance 3** Challenges students to write a program that detects obstacles and stops as their robot is traveling around a box.

**Big Idea 3: Make Sense of Systems**

Students are challenged to begin thinking about programming in terms of models and system. Three CYU questions follow the video and ask students:

- What kind of Model is the video talking about?
- What advantage does a “model based” understanding have over just memorizing facts?
- And, what is the “program flow” example in the video a model of?

**Lesson 6** Provides students with an opportunity to review working solutions of Mini Challenges: Square Dances 1 through 3. **The Container Transport Challenge** provides students with an opportunity to use a combination of feedback from multiple sensors, conditional loops, and movement commands to build a complex program that detects containers and moves them to a
loading zone. A quiz is recommended to assess students’ understanding of the Loops section.

**If-Else**

The If-Else unit is divided into 5 lessons and its challenge.

**Lesson 1** Introduces students to a robot designed to sort strawberry plants using If/Else logic. The included video also provides students with a preview of the robot challenges in the if/else unit. The video is followed by the following CYU question:

- What does an if/else statement enable your robot to do?

**Lesson 2** Provides students with the robot configuration that they will use for this unit: a driving base with both the ultrasonic and color sensors (facing forward).

**Lesson 3** Teaches students how to configure and use an if/else conditional block. The included video demonstrates how to program a robot that will move forward if there is nothing in front of it and will turn if there is something in front of it. Two CYU questions follow the video and ask students to explain how the robot makes decisions in the example video.

**Mini Challenge: Color Sensor Comparison**

Challenges students to use feedback from a color sensor to determine the robot’s behavior.

**Lesson 4** Introduces students to how to use a combination of if-else logic and loops to enable their robot to make decisions. A CYU question following the video asks students:

- What happens when you place an if-else conditional statement within a repeat forever block?

**Try It!: Smarter Maze Solver**

Provides students with the opportunity to revise their program to include usage of the front panel backlight. They are also challenged to change the robots behavior from repeating the behavior forever to repeating the behavior for a specific number of times.

**Mini Challenge: Tiles Maze**

Students are challenged to write a program that enables their robot to traverse a maze.

**Lesson 5** Provides an opportunity for students to review working solutions of the following Mini Challenges: Color Sensor Comparison and Tiled Maze.

**Strawberry Plant Sorter Challenge**

The Strawberry Sorter Challenge provides practice for students to use a combination of loops, if-else blocks, and repeat blocks to design a complex system that simulates the strawberry sorter. A quiz is recommended to assess students’ understanding of the If-Else section.

**Repeated Decisions**

The Repeated Decisions unit is divided into 5 lessons and its challenge.

**Lesson 1** Students are reintroduced to the autonomous tractor but this time they will learn how to have the tractor navigate through the orchard and make decisions at the same time. The included video is followed by a CYU question that designed to check students’ understanding of the programming problem.

**Lesson 2** Provides students with the proper robot configuration for this section: the driving base with both the ultrasonic sensor and the color sensor facing forward.

**Lesson 3** Introduces students to potential solutions to the autonomous tractor detection problem and describes why the solutions fail. This challenge is designed to help students critically think about program flow. Three CYU questions follow the video and ask students:

- Why doesn’t each of the following program work? (They are given the code to study)
- Instead of using a "waiting" approach, how do you think the problem should be
solved?

Lesson 4 Reviews the problems identified in Lesson 3 and then re-frames the task into a series of smaller steps that can be programmed using a combination of if-else statements and loops learned in the previous sections. Three CYU questions follow the video and ask students:

- What is a better way to think about obstacle detection?
- How do continuous decisions allow the robot to watch both sensors at once?
- How many times does the program process feedback from the if/else statement in the final solution?

Mini Challenge: Obstacle Detecting Until Black Line Challenges students to modify the Obstacle Detection program and create a program that will enable their robot to move until the color sensor detects a black line.

Lesson 5 Enables all students to review working code from the Obstacle Detecting Until Black Line Mini Challenge.

Obstacle Orchard Challenge

The Obstacle Orchard Challenge asks students to program the robot to move from its starting area through three rows of fruit trees, passing along both sides of each row. Again, this is similar to the Orchard Challenge but the added difficulty level here is the inclusion of randomly placed obstacles that the robot needs to react to. A quiz is recommended to assess students’ understanding of the Repeated Decisions section.

Line Tracking

The Line Tracking unit is divided into 4 lessons and the section challenge.

Lesson 1 Introduces the AMTS (Automated Material Transport System) which is designed to move parts at a manufacturing plant by using lines on the assembly room floor. Students are also introduced to the challenge in this section – to program a line-tracking robot. Two CYU questions follow the video and ask students:

- What environmental features will the robot use to navigate in this challenge?
- What programming concept will be used to create this behavior?

Lesson 2 Provides students with the proper robot configuration for this section: the driving base with the color sensor (facing down).

Lesson 3 Teaches students how a robot tracks a line and how to configure a lineTrack block. Three CYU questions follow the video and ask students:

- If a robot’s color sensor is seeing black what can it tell about its position?
- If a robot is tracking the left side of a line and sees black, which way should it move?
- Which basic program pattern is being used in a line tracking behavior?

Try it! 1: Right-edge Line Tracking Provides an opportunity to modify their program from a left line tracking behavior to a right line tracking behavior.

Try it! 2: Line Tracking Threshold Teachers students that while line tracking the color sensor is in a special gray-scale mode. The students’ task is to find the proper threshold value for the color sensor. Two CYU questions follow this Try it! and ask students:

- While line tracking, if the color sensor is giving a value above the threshold value what does that mean?
- If your robot is programmed to track the right side of the line, if the value the color sensor sees is below the threshold value which way should the robot move?

Try it! 3: Turning Ratio Challenges students to explore the effects of varying the motor speeds included in the lineTrackLeft and
The Virtual NXT is not a LEGO® MINDSTORMS® product. LEGO Education or the LEGO Group does not sponsor, endorse, or support this product.

### Mini Challenge: Line Track for 4 Rotations
Challenges students to write a program that tracks a line for a controlled distance. As an optional activity, students are then presented with a Did You Know?: How the Line Track Command Blocks Work which illustrates for students how the lineTrack command blocks contain a conditional block that can also be created using if-else conditional blocks.

### The Capstone Search and Rescue Challenge
This chapter provides a challenging culminating activity that is designed to incorporate all of the programming lessons that students have learned to date. Until now, most challenges have been simple and didn’t require significant student planning. This challenge provides an opportunity for students and teachers alike to look at the problem, break it into small parts, develop a flowchart that maps the program flow, and then solve the search and rescue challenge.

### Chapter Overview: Search and Rescue
In the Search and Rescue Unit, students will be required to develop flowcharts, they will learn an iterative approach to developing working robot programming code, and will learn two other Big Ideas: Break down problems and build up solutions, and Computational thinking applies everywhere.

Specifically, the Search and Rescue Unit covers the following topics:
- How to read and use flowcharts
- How to read and use action blocks
- How to read and use decision blocks
- How to read and use loops
- Practice breaking large programming problems into small behaviors
- How and why to use an iterative process when programming robots
- How to implement efficient project planning
- The importance of breaking down problems and building up solutions
- That computational thinking applies everywhere

### Lesson 1
Contextualizes the Search and Rescue problem by presenting a video that explains the roles of robots in emergency, hazardous, and disaster situations. After introducing that larger context, the video then explains what students will be required to program when they solve the Search and Rescue Challenge.

### Lesson 2
Reintroduces the Strawberry Sorter, only this video explains how the sorting robot makes its decisions. The decision making process is explained in the form of a flowchart and students learn how action blocks, decision blocks, and loops are used
to visualize program flow. Two CYU questions follow the video and ask students:
• What is a flowchart?
• Why are flowcharts important?

**Lesson 3** Provides students with a video that shows them how to iteratively test and improve their robot code. A CYU question follows the video and asks students:
• What is the recommended method for building and testing a programming solution?

**Big Ideas 4 & 5**

This video explains Big Ideas 4: Break down problems and build up solutions, and Big Idea 5: Computational thinking applies everywhere. Three CYU questions follow the video and ask students:
• What does it mean to “Break down problems”?
• What does it mean to “Build up Solutions?”
• What is Computational Thinking?

**The Search and Rescue Challenge**

The Search and Rescue Challenge is explained in detail. Please note that this challenge is considered to be the Capstone Challenge because it asks students to apply all of what they have learned. Below the video, there is a descriptive Challenge Overview and a PDF for the challenge that breaks it down into two phases. Although higher-ability students/classrooms are likely to succeed in the challenge when it is broken down into two phases, other classrooms might benefit from having the Capstone Project broken down further.

**Operation Reset**

This is a multifaceted game that includes multiple levels, a map, the ability to store in-game progress, and multiple robots for students to program.

The back story for the game is that a mining colony on Alpha Base H99 needs your help! A terrible storm has damaged the colony’s equipment and we need you to use your programming skills to restore communication systems, collect valuable Unobtainium Crystals, and refuel the Enigma Rocket so it can deliver them to Earth.

**Required:** Although the game has much more depth than what is suggested here, at a minimum students should be required to:
• Restore power to the three Communication Towers on the lower level of the colony.
• Collect at least 1 Unobtainium Crystal
• And, collect 3 Fuel barrels.

**Instructions:** Download and install the Operation Reset virtual world. Use the link to the right of the video to launch ROBOTC and the correct robot virtual world.

Pictured on the left are some of the printable resources available for students playing the Operation Reset programming game.
Programming is Precise

One of the big ideas of the Movement Chapter is that “programming is precise”. It will be important for students to change the way that they think about providing directions. The following activity will highlight the attention to detail required for students to become successful programmers.

The Humanoid Sandwich Programming Challenge

This activity requires students to program a human robot to make a two ingredient sandwich. This activity provides students with an opportunity to practice breaking down what appears to be a very simple task into its smallest parts. Students will quickly learn that they need to use very precise commands when programming robots. Making a two-ingredient sandwich (plus bread) is presumably familiar to most students, and that experience will allow them to think critically about the behaviors needed to program a humanoid robot. Initially, the majority of students will not be prepared to break this task down to a level of detail necessary for robot programming.

Procedure

The teacher says:

Your task is to program a humanoid robot to make a two ingredient sandwich. Begin by making a list of behaviors your robot needs to perform make a sandwich, then turn that list into pseudocode. Pseudocode is a simple set of instructions that you want the robot to execute. We will test your “code” by programming a human to execute your sandwich making program.

Students can begin writing their program individually or in small groups.

Identify one student to assume the role of the robot, they will stand in a place where everyone can see her/him. The props for making the sandwich can be actual bread, spreads/meats, and a utensil - or they can be stand-ins (paper for bread and/or meats, a pencil for a knife, cups for jars of spreads). The “robot” (student) will do exactly what fellow students say to do.

The teacher can ask the class for the first behavior to be programmed, followed by the second, third, etc. Most times, it quickly becomes apparent that students have not fully considered the level of detail required for programming. For example, the command “pick up a slice of bread” is inadequate. The robot needs to know in which direction to move to get the bread, how to detect the bread (use of sensory data), how to pick it up, etc. Those are each individual lines of code. It is not important to continue this exercise once students recognize why greater detail is required - even for a task as simple as making a sandwich.

Additional Lessons

This teacher’s guide provides multiple additional lessons that the teacher can use to introduce students to pseudocode and flowcharts. It is up to the discretion of the teacher to use or not use these lessons. For additional lessons go to the Index and find:

• Breaking Programs into Behaviors - page 44
• How does a robot think? - page 45
• Pseudocode and Flowcharts - page 48
• Introduction to Pseudocode - pages 49-52
• Pseudocode Exercise - page 53
• Introduction to Flowcharts - 54-56
What Are Behaviors?
A behavior is really anything your robot does: turning on a single motor is a behavior, moving forward is a behavior, tracking a line is a behavior, navigating a maze is a behavior. There are three main types of behaviors that we are concerned with: complex behaviors, simple behaviors, and basic behaviors.

Complex Behaviors
These are behaviors at the highest levels, such as navigating an entire maze. Though they may seem complicated, one nice property of complex behaviors is that they are always composed of smaller behaviors. This means that if you observe a complex behavior, you can always break it down into smaller and smaller behaviors until you eventually reach something you recognize.

Simple Behaviors
Simple behaviors are small, bite-size behaviors that allow your robot to perform a simple, yet significant task, like moving forward for a certain amount of time. These are perhaps the most useful behaviors to think about, because they are big enough that you can describe useful actions with them, but small enough that you can program them easily from basic EV3 ROBOT Graphical Commands.

Basic Behaviors
At the most basic level, everything in a program must be broken down into tiny behaviors that your robot can understand and perform directly. In the ROBOTC software, these are behaviors the size of single command blocks, like turning on a single motor, or checking a single sensor port. While these basic behaviors are very specific and immediately recognizable, they are not always terribly useful for programming, because they are often too small, and may not even be visible in the program’s output because they are too small.

Exercises
1. What level of behaviors can your robot perform directly?
2. Why is it useful to think about a robot’s actions in terms of behaviors?
Sense Plan Act

How does a robot think?
One easy to understand how a robot thinks is **Sense-Plan-Act**. A robot must be able to Sense its environment, Plan a course of action based on that data, and Act on that plan.

**Sense**
Using a variety of available sensors, the robot gathers data from its surroundings. Sensors include anything that provides the robot with information on its environment, such as the color sensor mounted on the robot in the picture, which will provide feedback about the color of the blocks in front of it.

**Plan**
The robot will process the information gathered in the Sense phase, and formulate an appropriate plan of action to react to what it saw. This step is most often performed by software (like your ROBOTC software) that has been loaded onto the robot in advance. The program illustrated here tells the robot to go forward until it sees a color.

**Act**
The robot acts in the world through the use of actuators—any component which allows the robot to create a change in its surroundings, such as motors, which move the robot through the environment. The robot in the picture needs to navigate the Orchard Challenge.

Answer the following questions

1. Define what a robot does.
2. Describe how your robot senses, plans, and acts to solve the challenge that you are working on.
Teaching How to Troubleshoot Programs

Stop, Trace, Analyze, Revise (STAR)

All teachers want students to learn how to troubleshoot their robot problems. There are times when a teacher will quickly give a student the answer, but this doesn’t teach them how to troubleshoot their own programming problems, it teaches dependency. If a teacher immediately answers all student questions, the student quickly learns that all they have to do is ask the teacher and their problem will be solved. We recommend the STAR approach to teaching troubleshooting, this approach is designed to teach students stop and trace the program flow, analyze where there is a difference between what should happen and what happened, and then revise and test the program.

Teacher’s Role in STAR Troubleshooting

Stop and Determine Student Intent The teacher needs to determine what does the student think is happening in the program. The teacher needs to ask the students to:

“Please describe step-by-step what is happening in your program”.

Trace Together, the student and teacher, need to trace through the program and identify where the robot’s behavior diverges from the student’s intent.

Analyze The teacher needs to help the student to analyze what they misunderstood.

Revise Help the student to correct their misunderstanding and to fix their program. Often, correcting a student’s misunderstanding means re-framing the problem or highlighting some discrepancy in how the student sees the problem.

STAR - Student’s Role in Troubleshooting Programs

Stop and Reflect What do I think should be happening in each step of my program?

Trace I need to trace through the program step-by-step and identify what my program is telling the robot to do. If it is a complex program, then I should develop a flowchart that allows me to see my program flow.

Analyze If the robot is misbehaving, I need to analyze what parts of the program works and identify the point where the robot stopped doing what I wanted it to do. Then I need to figure out what I need to change to fix the problem

Revise I need to scientifically correct the program one step at a time and test each part of the program and fix the problem.

Note: Teachers might choose to post the student version of STAR somewhere in their classrooms and direct students’ attention to it as they begin asking for assistance.
STAR Troubleshooting Power Point Slides

The Movement, Sensors, and Program Flow Chapters each contain a set of “STAR Troubleshooting Power Point” slides that a teacher can use with the whole class to introduce the STAR method of troubleshooting. See the Power Point folder in the EV3 curriculum to review the slides. Each slide contains a common misconception that some students might have involving programming and/or logic. The Power Point slides are designed to be used as an anticipatory set and used during whole class discussions at the beginning of the period. The slides are designed to be used at the teacher’s discretion.

Each slide asks students to identify what they think that programmer’s intention was and to troubleshoot the code and fix it. The teacher’s role is to ask students questions designed to help students learn program flow as well as additional troubleshooting techniques. The slides present opportunities for the teacher to check student’s understanding of individual command blocks as well as ask students how they might test their new code to insure that it works.

Movement Power Point Slides

This slide set contains 13 slides. It includes one slide that highlights the Compiler Target. The remaining slides are designed to check a student’s basic understanding of the forward, turnLeft, turnRight, and moveMotor command blocks.

Introduction to Sensors Power Point Slides

This slide set consists of 16 slides. The teacher will need to sort through the slides and use them as a particular sensor is being introduced. For example, there are slides on the Touch, Gyro, Distance, and Color Sensors and the slides will only be appropriate if that particular sensor has been introduced to the student. New EV3 teachers should test the programs before you use the slides so that you understand both the intent of the slide as well as what the program will actually do. There are nine slides for Forward Until Touch, two for Forward Until Near, five for Turn for Angle, and two for Forward Until Color.

Introduction to Program Flow Power Point Slides

This slide set consists of 20 slides. Once again, the teacher will need to sort through the slides and use them as a particular Program Flow concept is being introduced. For example, there are slides on the Line Tracking, If/Else, and Repeated Decisions in the slide set and these concepts may not have been introduced yet. New teacher’s should test the programs before you use these slides so that you understand both the intent of the slide as well as what the program will actually do. There are four slides for Loops, seven slides for If-Else, three slides for Repeated Decisions, and three slides for Line Tracking, and three general slides.
Introduction

**Pseudocode** - is a native language description of what the robot is required to do. With practice pseudocode eventually resembles ROBOTC code.

**Flowchart** - is a graphical representation of program flow.

It is a very good practice to have students begin each programming task by breaking the task into its smallest parts and develop pseudocode that describes the robot’s behaviors. You will find much more detail on this practice in the *Introduction to Pseudocode* lesson.

The practice of developing pseudocode is informally introduced on the first day of class when students are asked to write a program to control a humanoid robot to make a sandwich. See *Introduction to Robot Programming* for a full description of that activity. In that activity, students will not yet know a robot programming language, but they can begin to write pseudocode to describe the behaviors the robot will need to complete to make the sandwich. (E.g., lift the arm, open the hand, reach for the bread, grab the bread, etc.).

In this teacher’s guide, we use the term *flowchart* in a very general way to convey that students are constructing visual representations of the problem and its solution. The advantage to flowcharts is that they help a student identify robot decision making.

This curriculum doesn’t formally introduce flowcharts until the Program Flow Chapter because the first two chapters, Basic Movement and Sensors, only require simple behaviors (movingForward, turning, moveUntil a sensor value, etc.). During these activities, students will learn the basic lexicon of the programming language allowing them to begin writing robot behaviors using pseudocode. Teachers should encourage students to use pseudocode - even if it initially starts as common descriptions and slowly improves to reflect a more accurate programming language. Pseudocode will eventually be used to describe individual parts within a flowchart since each pseudocode phrase describes a unique robot behavior.

In the Program Flow Chapter, students will begin programming their robots to solve problems that contain multiple steps with decisions, that is when this curriculum formally introduce flowcharts. Additionally, you will find an activity called Tic-Tac-Toe which can be used to help students learn to think algorithmically. Students learn are tasked to write a tic-tac-toe to break down the game into its basic parts and then describe the algorithm used to play the game. This can be a group activity centered around developing a flowchart to determine how to play the game. With practice, students will begin to see that flowcharts provide a visual representation of the robot’s decision making process and then begin using flowcharts to solve their programming challenges.
Overview

In order to plan a program and write efficient code students need to be able to write clear instructions for the robot. This lesson introduces students to the idea of writing clear instructions and then introduces them to pseudocode.

Objectives

Students will be able to:
• Listen carefully and follow instructions
• Communicate clear instructions
• Break tasks down into smaller pieces
• Write pseudocode for a simple maze
• Understand the necessity of planning clear steps

Materials

• Blank paper for each student
• Pencil and a ruler for each student
• A large table top or floor surface to setup a simple maze
• Tape to mark out maze boundaries
• Small box or object to represent a robot

Clear Communications Lesson

Procedure

1. Introduce the following “Clear Communications” drawing activity. Tell students:

Robots will follow the program that they are given, even if that program doesn’t make sense. The robot needs to be given a program that it can understand to produces the desired outcome. Let’s begin with a drawing activity to help us think about this idea of giving clear instructions.

Have all students begin with a blanks sheet of paper and then give them the following verbal instructions:

1. Draw a dot in the center of your page
2. Draw a vertical line from the top of your page to the bottom of your page, passing through the center dot
3. Draw a horizontal line from the top of your page to the bottom of your page, passing through the center dot
4. Write the word ROBOT center of the square created at the bottom right of your page

Have students hold up their drawings when everyone is finished. Check to see that everyone’s is the same. Discuss any differences if they arise.
Introduction to Clear Communications

Choose a student to be the communicator and give them a simple drawing (see examples at the bottom of this page) making sure no one else sees the picture that you’ve given the student communicator. Have the communicator describe the drawing for the others to reproduce on their paper. The other students may ask questions for clarification and the communicator may adjust their instructions if they see mistakes being made. Encourage students to see how quickly and accurately the picture can be reproduced – each drawing should not take more than 5 minutes.

Repeat the exercise with a new picture and a new communicator student. This time the other students are not allowed to talk or ask any questions, but the communicator may still adjust their instructions if they see mistakes being made.

Repeat the exercise one last time with a new picture. Now the communicator must sit behind a screen or with their back turned and no questions may be asked. The communicator must clearly give their instructions one time for the others to reproduce the drawing.

Discuss this last exercise with the class.
Ask the students who had to be the communicator:
How did you decide the order in which to give instructions?

How confident were you that your instructions would work?

Ask the students who were reproducing the drawing:
Did you find the instructions easy to follow?

Was there anything you drew differently because the instructions were unclear?

What makes a good set of instructions?

[Emphasize this question, and encourage students to conclude that the instructions need to be broken down into simple steps because instructions that are too big can be confusing and lead to error.]
**Introduction to Robot Programming**

Tell students: *This last exercise is most like programming robots – the programmer (or communicator) gives a set of instructions which will be followed exactly. The instructions need to be broken down into the simplest possible pieces so that they can be performed accurately and without confusion.*

Tell students: *Behaviors are a very convenient way to talk about what a robot is doing and what it must do. Every action of a robot can be described as a behavior: move forward, turn on a motor, look for an obstacle, stop, and solve a maze are all examples of robot behaviors.*

Direct students to the simple maze shown below on the left. The goal is for the robot to solve the maze by following the path below on the right. Tell students: *To do this, we will think about the robot’s actions in terms of behaviors.*

![Simple Maze](image1)

![Simple Maze Solution](image2)

Give a student a robot. Have a student volunteer move the object along the path that would solve the maze. Point out that we can easily see what behavior the robot needs in order to solve the maze.

Tell students: *Some behaviors are too big to give to the robot as instructions, so we need to think about breaking them down into smaller behaviors. “Solve the maze” is actually a very complex behavior because it involves many steps and the robot wouldn’t know what to do.*

![Simple Maze Breakdown](image3)
Instruct students to:
*Write down the behavior “Solve the maze” on the top of your paper.*

Now, tell them to:
*Break down that behavior into smaller behaviors and write them below in order.*

If students are unsure, ask:
*What does the robot need to do to follow the solution path?*

(Move forward, turn left, move forward, turn right, etc.)

Ask students:
*Is your list of behaviors is clear enough to instruct the robot through the maze.*

They should realize that we are close, but the robot doesn’t know how far to move forward each time or how much to turn.

Finally, instruct students to:
*Use your rulers to figure out the distance the robot needs to move for each behavior.*

(Exact distances will depend on your maze setup).
*Write this new specific behavior next to their last list of behaviors.*

We now have a list of behaviors specific enough to give as instructions to the robot.
Tell students:
*By starting with a very large solution behavior and breaking it down into smaller and smaller sub-behaviors, you have a logical way to figure out what a robot needs to do to accomplish its task.*

Talking about and writing the code in English is the first step in good pseudocode practice, which allows us to plan robot behaviors before we translate them to code.
Pseudocode Exercise

What is Pseudocode?

Robots need very detailed and organized instructions in order to perform their tasks. Before a programmer can begin programming they need to break a robot’s behaviors down into simple behaviors and figure out when each behavior should run. Some programmers like to use pseudocode to begin ed-constructing the programming problem.

pseudo

adj : not genuine but having the appearance of;
Source: WordNet ® 1.6, © 1997 Princeton University

Pseudocode is a hybrid language, halfway between English and code. It is not real code yet, but captures the details that will be important in translating your ideas to code, while still allowing you to think and explain things in plain language. Good pseudocode will make it very straightforward to write real code afterwards, because all the behaviors and logic will already be contained in the pseudocode.

Pseudocode example

If you wanted to program a robot to stop when it saw an object and move forward when it didn’t see an object your pseudocode might look like:

**pseudocode**

1. Move forward
2. If (sonar sensor detects an object)
   stop
3. When the sonar sensor no longer
   sees and object move forward.
4. Do this forever

Exercise

1. Convert these instructions to pseudocode and into a flowchart:
   a. “If it’s raining, bring an umbrella.”
   b. “Keep looking until you find it.”
   c. “Take twenty paces, then turn and shoot.”
   d. “Go forward until the touch sensor (on port 1) is pressed in.”
   e. “Turn on oven. Cook the turkey for 4 hours or until meat thermometer reaches 180 degrees.”
   f. “Crossing the street” Hint, make sure that you look both ways!

2. Compare the advantages and disadvantages of flowcharts and pseudocode.
   Explain in your own word why you believe one is better than the other.
   Is one of them always better than the other, or are both good in different situations?
   Can you use both to help solve the same problem? Should you?
Overview

A flowchart is a visual representation of program flow and is used by programmers to break down and model robot behaviors. This lesson provides teachers with a guide to introduce flowcharts in the Sensors - Forward Until Touch Unit, and then this lesson should be applied to all of the other Sensor Units. Students have already been introduced to pseudocode and program flow in the Introduction to Robot Programming activity; this lesson focuses on how to graphically describe robot decision making. In this lesson the teacher will model how integrate pseudocode and decision making into a flowchart.

Objectives

Students will be able to:
• Break down a problem into simple components
• Organize the components into a proper sequence
• Represent a sequence of behaviors in a flowchart
• Use a flowchart to analyze robot behaviors

Procedure:

Have students watch the video and complete the questions in the Sensors - Forward Until Touch 3: Move Until Touch Unit. In this example students will write a program that requires the robot to make a decision and includes multiple steps. This is a good time to introduce flowcharts. Note: All students should be able to see a picture of the flowcharts below.

Tell students: Now that the robot is using a sensor to make decisions, we can use a flowchart to understand how our robot’s behavior is broken down into steps within a program. So far, our flowcharts have just been single steps of pseudocode, but now we will need to add decision blocks which ask a “yes or no” question.

Show both of the following flowcharts to the class and ask:
Which flowchart best represents the robot behavior seen in the Move Until Touch video?

![Flowchart 1](#)

![Flowchart 2](#)

Draw the example flowcharts on the left on the board or project them onto the screen for students to see and discuss.
If students struggle or disagree, ask:

*When does the robot stop moving forward?* *(When the bumper sensor is pressed in.)*

*When does the robot check the bumper sensor to make the decision to move forward or stop?* *(The robot continually decides to move forward when the bumper is not pushed, and decides to stop when the bumper is pushed)*

*Which flowchart represents a robot continually making decisions based on the bumper sensor?* *(The flowchart on the right requires the robot to keep asking if the bumper is pressed)*

Read through the flowchart on the right (on the previous page) to the class to show how each robot action is represented in the flowchart. Teach students the difference between start/stop, action, and decision blocks.

Tell students: *The robot must continually check the value of the bumper sensor to decide what to do. If the bumper sensor is not pressed, the robot continues running both motors forward. If the bumper sensor is pressed, the robot stops all motors. The flow of these decisions is given in the flowchart on the right, but missing from the chart of the left."

Go to **Forward Until Touch 3 -Try it! 2** activity. Refer to the same flowchart as the class thinks about the following question:

*What happens if you start the program with the bumper switch pressed in?"*

Give them a moment to think, and then ask students to explain their answer to the class using a flowchart. Tell the students: *Use a flowchart to break down each step of the robot's behavior so that we can see what actions the robot will perform in any situation.*

Next, scroll down to the **Forward Until Touch - Try it! 3** activity. Take down the previous flowchart, and show this new flowchart with missing entries. Have students copy it onto their papers and fill in the blanks to properly represent program flow while a robot is pushing a box until its touch sensor is released.

If students struggle, ask: *Now when will the robot stop moving forward?* *(When the bumper sensor is released)*

When students have successfully filled out their flowcharts, ask the class:

*What would happen if the box was not all the way against the bumper sensor when the program started?* *(The robot would not move)*

Use a flowchart to show this pathway.
Scroll down to **Forward Until Touch - Mini Challenge 1: Vacuum**. On a new piece of paper, have students design a flowchart to represent the robot’s behavior in this challenge. The robot will need to perform more actions after the bumper sensor is read each time.

Give the students several minutes to work on their flowchart. Have them work in small groups if necessary.

If students struggle, ask:
- *What does the robot need to do after the bumper sensor is pressed?* (Backup and turn towards another wall)
- *Does the robot ever need to repeat its behaviors?* (Yes, it will have the same behavior for all 4 walls)
- *How will the robot know when it has contacted all the walls?* (When it has repeated its behavior 4 times)

After students have finished, ask a group to draw their solution for the class. Follow the flow and check the logic for any missing steps or problem areas.

Ask if other students created a different flowchart they would like to share.

Discuss aspects of each solution until the class comes to an agreement on a correct flowchart. It’s possible to have multiple correct solutions.

Tell students:
*Flowcharts help us understand the decision making process of a robot. The answer to a “yes or no” decision will determine what action the robot does. By creating a properly organized flowchart, we can see the plan a robot needs to follow to be successful.*

At this point, either have students use their flowcharts to write the code and try each challenge, or move on to the tic-tac-toe activity to transfer the critical thinking.
Flowchart Exercise

What are Flowcharts?

Robots need very detailed and organized instructions in order to perform their tasks. The programmer must break things down into simple behaviors and figure out when each behavior should run. A flowchart is a tool that can be used by programmers to determine program flow.

A flowchart provides a way of visually representing and organizing individual behaviors and decisions within a program -- it provides a diagram of the “flow” of the program. Programmers use flowcharts to lay out the steps that will be needed in their final program, and to help determine how the robot’s behaviors should be broken down.

Parts of a Flowchart

- **Start of Program** - Marks the beginning of the program, begin here. Follow the line to get to the next block.
- **Statement Block** - A statement to execute, or a behavior to perform.
- **Decision Block** - A decision point in your program. Ask a simple question, and do different things depending on the answer.
- **Yes/No** - Answers to the question posed in the decision block. Follow the line labeled with the appropriate answer.
- **End of Program** - Marks the end of the program. If you reach this point, the program is done!

Exercise

1. Make a flowchart organizing the “flow” of getting ready to go to school in the morning. Be sure to include the following steps in your chart, but don’t be afraid to add other things if you need them!

<table>
<thead>
<tr>
<th>Select something to wear</th>
<th>Look for your shoes</th>
<th>Put your shoes on</th>
</tr>
</thead>
<tbody>
<tr>
<td>Take a shower</td>
<td>Brush your teeth</td>
<td>Hit snooze button</td>
</tr>
<tr>
<td>Eat breakfast</td>
<td>Put toast in the toaster</td>
<td>Get dressed</td>
</tr>
<tr>
<td>Walk or get a ride to school</td>
<td>Check your alarm clock</td>
<td>Comb your hair</td>
</tr>
<tr>
<td>Get out of bed</td>
<td>Turn on shower</td>
<td>Check the time</td>
</tr>
</tbody>
</table>
This teacher-led activity helps students to extend their computational thinking to an everyday problem. In this activity, students are asked to develop an algorithm to play tic-tac-toe. Teachers will find a prepared slide presentation designed to help students think about solving the game. The game is a simple game that most students quickly find that there is no way to win. To ensure that all students are familiar with the game, have them play several games with each other and discuss their strategy before beginning the activity.

The following is the suggested discussion surrounding the introduction to the tic-tac-toe programming presentation, teachers will find a Power Point Presentation in the teacher resources folder:

Slide 1: The teacher says:
*The algorithmic thinking skills that we’ve been developing can be applied to problems outside of robotics. Let’s take a look at how we can use what logic to develop a strategy for winning at tic-tac-toe.*

(Teacher can read through the big ideas of the unit with examples of how each is applied to this problem.)

*Let’s start by considering simple aspects of playing tic-tac-toe.*

*How many ways can you win horizontally?*

(Discuss the term, horizontally, if necessary.)

Slide 2: The teacher says:
*Here we see the three ways a player can win horizontally.*

*How many ways can you win vertically?*

(Discuss the term, vertically, if necessary.)

Slide 3: The teacher says:
*Good, and here are the three ways to win vertically.*

How many ways can you win diagonally? (Discuss term, diagonally, if necessary.)

Slide 4: The teacher says:
*Right, there are only two ways to win diagonally.*

*So how many ways is there to win total?*

Slide 5: The teacher says:
*Right, 8 ways total. Let’s organize and label those ways in an array or grid like the one used for tic-tac-toe.*

Slide 6: The teacher says:
*Here’s the nine-boxed array like the grid used in the game. You can see that each possible way to win is labeled. If I’m the first player, how can I decide where to put my X first?*

Continued next page.
(Students might respond in a variety of ways but they should eventually recognize that as the first player, the X should be placed where it provides the greatest number of opportunities to win. This is explained to students on the next slide.)

Slide 7: The teacher says:
*Right, considering which box will provide the most possible wins is a good strategy.*

Slide 8: The teacher says:
*Here we can see that the center box has the greatest number of possible wins - 4.*

*Which are those four possible ways to win?*

Slide 9: The teacher says:
*Right, here we see those four ways.*

*So we place our X in the middle square.*

**The Student Challenge**

Students are now challenged to create a flowchart that documents the decisions involved when deciding where to place the first X in a game of tic-tac-toe. The flowcharts will include pseudocode that describes how they would program a robot to play tic-tac-toe.
Movement - Straight

Moving Straight Reflection Question

In what ways are the following two programs alike? In what ways are they different? Some smaller differences are circled in red to make them more noticeable.

Program 1:

```plaintext
1  forward (1, rotations, 50);
2
```

Program 2:

```plaintext
1  setMultipleMotors (50, motor1, motor6, , );
2  waitUntil (getMotorEncoder(leftMotorNL) == 360);
3  stopAllMotors();
4
```

Please write your answer below:
Shawna believes her robot is broken. She built an EV3 robot, and wrote the following program to make the robot turn its body one-quarter (0.25) of the way around:

```
1. turnLeft (90, degrees, 50);
```

However, it does not actually turn that much when it is run. Should Shawna’s robot be replaced because it is broken? Explain why or why not.

Please write your answer below:
Teacher note - The reflection question at the end of each chapter could be handled as a written student homework assignment or could be an in-class discussion. It is at the discretion of the teacher how he/she handle the reflection question in their classroom.

Turning Reflection Answer Key

No, the robot should not be replaced for the reason claimed. It is a common student misconception that when a student enters 90 degrees into the TurnLeft command block that the robot will make a 90 degree turn.

In reality, the number entered in the first number block of the “TurnLeft” command block sets the percentage of rotations that the robot’s wheel will turn. 90 degrees, therefore, means that the robot will move in a turning motion until its wheels have completed 90 degrees of rotations of the robot’s drive wheel. The exact rate at which motor rotations produce robot body rotation depends on the robot’s gearing and the size and physical arrangement of the wheels.
EV3 Turning Investigation

Investigation description

Controlling robot actions and making precise movements is an important part of robotics. In this investigation we will look at making accurate turns using different conditions.

**Point turn** – in this type of turn one drive motor is powered forward and the other drive motor is powered in reverse causing the robot to turn on its central axis.

**Swing turn** – in this type of turn the drive motor on one side of the robot is powered and the other side remains still. (For our investigation we will only be using point turns.)

Procedure

Write a program that allows your robot to execute a 90 degree point turn. Notice that there are 5 different options in ROBOTC for turn commands: degrees, rotations, milliseconds, seconds, and minutes.

Iteratively test and adjust your program until you find the right values to make a precise 90 degree turn using rotations, degrees, and seconds. Use the data tables on the next page to record your results.

After you have completed the 90 degree turns, use your results to calculate a value for making precise 45 degree and 180 degree turns. Test your calculated values and then use the gyro sensor value or a protractor to determine the actual degrees turned. Calculate the percent error using the equation given.

Reflection

After you have completed the data tables, use a separate sheet of paper to write complete answers to the following prompts.

1. Did motor power have an impact on your value for a 90° turn using rotations and degrees? Why?
2. Did motor power have an impact on your value for a 90° turn using seconds? Why?
3. Did your calculated values work for producing precise turns? Or did your robot turn more or less? Why?
4. Write a one-paragraph conclusion summarizing your results. Be sure to include data for support.
Complete the data tables below. Be as precise as possible with your measurements.

### 90 Degree Point Turn

<table>
<thead>
<tr>
<th>Rotations:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motor Power</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>60</td>
</tr>
<tr>
<td>90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Degrees:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motor Power</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>60</td>
</tr>
<tr>
<td>90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Seconds:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motor Power</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>60</td>
</tr>
<tr>
<td>90</td>
</tr>
</tbody>
</table>

### Testing other Point Turns

Use the following formula to calculate percent of error:

\[
\frac{\text{predicted degrees} - \text{actual degrees}}{\text{predicted degrees}} \times 100
\]

<table>
<thead>
<tr>
<th>Rotations:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turn Amount</td>
</tr>
<tr>
<td>45 degrees</td>
</tr>
<tr>
<td>180 degrees</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Degrees:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turn Amount</td>
</tr>
<tr>
<td>45 degrees</td>
</tr>
<tr>
<td>180 degrees</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Seconds:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turn Amount</td>
</tr>
<tr>
<td>45 degrees</td>
</tr>
<tr>
<td>180 degrees</td>
</tr>
</tbody>
</table>
CHAPTER 1: Sensabot Challenge

In this challenge, you will program your EV3 robot to move from its starting box to three different lines on a game board, stopping at each one to perform an inspection, represented by lowering and raising the robot’s arm. When the robot is done inspecting all three location, it should back up and return home to its starting box to recharge.

Rules and Procedures:

- Create the robot’s starting area with electrical tape that is slightly larger than the robot.
- Use the electrical tape to mark three (3) inspection points along the robots path. The exact location are not important, but they should not be moved once the board is finalized.
- Robot must start inside the starting box (no parts over the line) and with its arm raised.
- The robot must move and stop at each line, lowering and raising its arm, representing the inspection process. The arm must be directly over each line when the inspection is performed.
- The robot must return to its starting box after completing the inspection process at the third line. The entire robot must be inside the box (no parts over the line)

Hints:

- Use a meter stick or ruler to measure the distances to each line on the board so you know how far you need to move each time!
- Try finding the number of centimeters your robot travels in each rotation, and using that to find the number of rotations you need
- You can also make a test run, then calculate “how many times as far” you need to move to get to each line, compared to the test run
CHAPTER 2: Orchard Challenge

In this challenge, you will program the robot to move from its starting area through three rows of fruit trees. You may choose your own path through the orchard, but the robot must pass alone both sides of each row during its run.

Rules and Procedures:

- For this challenge, the user can create their starting area wherever on the board.
- Use three strips of electrical tape to mark three rows of trees. The exact locations are not important, but they should not be moved once the board is finalized.
- Make sure there is enough space between the rows for the robot to pass on both sides of each row without crossing the lines.

Hints:

- Use a meter stick or ruler to measure the distances to each line on the board so you know how far you need to move each time.
- Try finding the number of centimeters your robot travels or number of degrees its body turns in each wheel rotation.
- You can also make the test run, then calculate “how many times as far” you need to move or turn to get the amount of movement you want, compared to a test run.
CHAPTER 3: Arm Position Challenge

In this challenge, you will program the robot’s arm to move into the “Up” position when the “Up” button on the EV3 is pressed, no matter where the arm started. The robot will then move forward five (5) rotations to pick up a cargo container, and bring it back to the starting location.

Rules and Procedures:

- The robot’s arm will be moved to a random position before running.
- You must use parts available to you to modify the Touch Sensor’s mounting so that it can detect when the EV3RE’s arm is in the “up” position.
- Try to do this with as few changes as possible.
- When the “Up” button on the front the EV3 is pressed, the robot must raise its arm to the “Up” position (1), move forward two rotations to the cargo (2), and bring it back to the original starting position (3).

Hints:

- The trigger area on the Touch Sensor is small. You will probably need to build a “bumper” or “extender” on the end of the Touch Sensor to make it detect the arm more reliably.
- The EV3 core set includes two Touch Sensors. You can use one sensor for detecting the arm in the “Up” position, and the other to detect the box.
- You can also make a test run, then calculate “how many times as far” you need to move or turn to get the amount of movement you want, compared to a test run.
CHAPTER 4: Maze Challenge

In this challenge, you will program the EV3 robot to move from its starting area through a maze with tall, vertical walls. Walls in the maze can be adjusted to be nearer or farther between each run, but the general path must remain unchanged. Make use of the Ultrasonic Sensor and be sure to not touch any walls along the way.

Rules and Procedures:

- Recreate the maze with tall objects, such as books or binders.
- The robot must start entirely within the start box (1), as well as stopping entirely inside the end box (2).
- Walls marked by gray lines can move slightly.
- Walls marked by dark black lines cannot be moved.
- The robot must not touch any walls as it navigates throughout the maze.

Hints:

- The patterns of turns cannot change, so you don’t need to worry about using a sensor (other than Rotation) during turns.
- Look carefully at what distances in the maze are guaranteed not to change. Try to use those distances in combination with the Ultrasonic Sensor to orient yourself.
CHAPTER 5: Mower Challenge

In this challenge, you will program your EV3 robot to erase or clear the entire gameboard of either markings or parts. The robot is able to move freely in straight lines, using any method you want. However, there are three mud zones marked on the game board. When turning in one of these areas, the robot must be picked up by hand, and placed back down.

Rules and Procedures:

- The robot must clear the board of marks or parts to complete its mission.
- If any part of the robot is in the mud zone (red squares) at any point during a turn, it must be picked up and placed back down, as close to the same spot and facing as possible.
- Use overlapping paths to compensate for sensor inaccuracies.

Hints:

- Because the robot’s wheels continue to spin in the air when it is picked up, Rotations or Time will not be reliable for turns in the mud.
- The Gyro Sensor responds only to the robot’s body turning, and is unaffected by interruptions like being picked up.
- It is unlikely that the eraser (or scoop) will perform perfectly, especially near the edges of its reach. Plan your robot’s course accordingly.
- Use an adjustment factor to compensate for the fact that the robot won’t see 90 degrees until it is past 90 degrees.
CHAPTER 6: Traffic Signal Challenge

In this challenge, you will program your EV3 robot to through three different intersection, each of which has a traffic signal. The traffic signal, which can be either the colored block or the red/green card, is held by hand at a set height. Unlike a camera, the detection range of the Color Sensor is short, so you will need to modify its placement on the robot so that it can see the traffic signal and react appropriately.

Rules and Procedures:

- Traffic signals are represented by holding the colored block with the signal side downward and toward the robot at a 45-degree angle.
- You can also use red and green colored sheets of paper, which may be easier to hold at the correct height.
- The initial signal color for each intersection is determined by flipping three coins before the run. Heads = Green, Tails = Red.
- The robot MUST stop if the light is red, and MUST NOT stop if the light is green.
- After the robot successfully gotten through all three intersections, the robot can be stopped by hand.
- BONUS: Write the program so that it DOES always stop after going through the third intersection. This will require the use of a Switch.

Hints:

- Since you only have access to Wait commands, try to break the robot’s behavior into stages. What is the robot waiting to see at each stage of its movement? What should its motors be doing during those stages?
- If the robot is already moving, is there any point in waiting for Green? If the robot is stopped, is there any point in waiting for Red?
- If the robot sees a Red light, don’t forget that it needs to wait for the actual Green light before proceeding!
CHAPTER 7: Container Handling Challenge

In this Challenge, you will use a Loop to program your robot to move a series of containers into a loading zone. The containers to be loaded are placed at irregular intervals, so you will have to use a sensor to detect each one. The robot should then use its arm to transport the container back into the loading zone – marked with a red outline – and release it there.

Rules and Procedures:

- Use four rectangular objects to represent the four containers. These can be built with LEGO parts, or cut from cardboard boxes.
- The four objects must be placed in a straight line at the start of each run, with a flat side facing the robot, but their distance from each other should be varied randomly.
- The drop-off area for the containers should be marked with red electrical tape. If red is not available, any color may be substituted as long as it is different from the color of the table surface.
- A container with no parts hanging outside the loading area may be removed by hand to make room for the next container.
- The robot must return all four containers reliably to the loading area, regardless of where they were located along the initial line.

Hints:

- Use a Loop to repeat similar portions of the behavior. You should not have to write all four runs separately!
- If the robot needs to perform any actions prior to the loop, simply place them before the Loop in the program.
- If the robot needs to perform any actions after the loop, simply place them after the Loop in the program.
- Use Sensors to help the robot locate both the containers and the loading zone, as the exact distances required will be different each time the robot runs!
CHAPTER 8: Strawberry Plant Challenge

In this Challenge, you will combine both Loops and Switches to repeat an inspection process on a total of four plants, represented as boxes. The robot will move to a plant, then sort it to either the robot’s right side if it is good, or the robot’s left if it is bad.

Rules and Procedures:

- Use four rectangular objects to represent the plants. These can be built with LEGO parts, or cut from cardboard boxes.
- The objects must be placed in a straight line, with at least 5 centimeters’ space between them.
- The objects should be a random mix of “good” and “bad” plants, and placed in a random order for each run.
- Green or brown plants are considered “good”.
- Red, black, or yellow plants are considered “bad”.
- The robot must correctly sort all plants in a run to complete the Challenge.

Hints:

- Use a Switch to conduct individual inspections.
- Use a Loop to repeat the inspection process.
- According to the rules, when should the robot stop inspecting?
CHAPTER 9: Orchard Obstacle Challenge

In this challenge, you will program your EV3 robot to move from its starting area through three rows of fruit trees. In addition, however, there will be one or more obstacles placed at random throughout the orchard. The robot should not touch these obstacles; instead, when it encounters one, it should stop moving until they are removed by hand... at which point the robot should continue on its way.

Rules and Procedures:

• This challenge uses the same game board layout as the Orchard Challenge from Chapter 2 (Turning).
• Like the previous challenge, the robot can start anywhere there is space available.
• Place one to two obstacles randomly alongside a side of a row for the robot to encounter.
• Be aware to not place an obstacle where the robot may bump into when turning a corner.
• When the robot encounters an obstacle, it should stop and wait for the Obstacle to be removed by hand. It should then continue moving without additional human intervention.

Hints:

• Use a meter stick of ruler to measure the distances to each line on the board so you know how far you need to move each time.
• The obstacle can be completely removed from the challenge after the robot approaches it and stops.
• Use lower speeds to minimize of effects of momentum when turning.
CHAPTER 10: Line Track Challenge

In this challenge, you will program your EV3 robot to grab a cargo crate from the pickup spot, follow the line track and drop the crate off in the drop-off zone.

Rules and Procedures:

• Build the course shown above using black electrical tape on a light surface.
• The robot must pick up the box using its arm, then follow the line to the drop-off point, and release the box there.
• Program your robot to deliver the box in the shortest time possible!

Hints:

• You can use the Loop Mode setting to adjust when a Line Track behavior ends. The program will then run whatever comes next, even another Line Track!
• Use multiple line tracks with different “sharpness” one after another to handle parts of the board.
• Adjust the “sharpness” of the robot’s motions using the Steering slider on the Move Steering Block.
• You may find it advantageous to track the right side of the line in some places. Which way should the robot go when it sees Black to get to the right edge of the line? Which way should it go to get back after it drives off?
Search and Rescue Programming Challenge

In this Challenge, you will use everything you’ve learned to create a rescue robot that will enter a 4-room building. The robot must perform 4 unique actions for 4 unique rooms, that will be randomized in order to simulate a hazardous area where you can never know what will be encountered. The robot must complete all 4 rooms, and return to the starting point.

Final Challenge Board Setup

- The rooms’ basic shapes are all identical.
- The building walls do not need to move when the rooms are randomized!
- Each room either contains one of three props or contains nothing at all.

Room 1: Fire
A sizeable square of red electrical tape on the ground. For easier portability, you can attach most of the tape to an index card or paperboard, and only attach/detach the edges from the table.

Room 2: Rescue
A piece of PVC pipe, toilet paper tube, or small tower constructed of LEGO elements. A hook on the side of this object should be suspended about 3 cm above the table surface so that the robot’s arm can lift the person off the table.

Room 3: Walled
Any piece that blocks the robot’s travel. It can be a piece of tape, if nothing else is available. The robot must be able to get around the obstacle, so make sure that there is enough space for the robot to go around it.

Room 4: Clear
One room will not contain any props for this challenge.
PHASE 1

Before you build a robot that can complete objectives of all 4 rooms at once, demonstrate that your robot can complete each rooms, one at a time.

- Write 4 separate programs for each room
- Robot can enter either entrance of the room
- Robot must exit the room completing the objective

### Room Objectives

#### PROGRAM 1: Fire Room

Run over ‘Fire’ area with the rear end of the robot

#### PROGRAM 2: Rescue Room

Pick up the survivor

#### PROGRAM 3: Walled Room

Avoid the walled area and exit

#### PROGRAM 4: Clear Room

Play sound ‘Analyze’ before exiting the room
**PHASE 2**

Combine your robot’s capabilities in Phase 1, and build a robot that can distinguish and complete all 4 rooms in one run.

- Write 1 program that will travel all 4 rooms
- The location of the rooms will be randomized each run
- The robot may start at any room’s entrance
- The robot’s trip can be either clockwise or counter-clockwise
- The robot must return to where it started

---

**Final Challenge Objective**

---

![Diagram of robot and rooms]
CHAPTER 1: Sensabot Robot Virtual World Challenge

In this challenge, you will program your EV3 robot to move from its starting box to three different lines on a game board, stopping at each one to perform an inspection, represented by lowering and raising the robot’s arm. When the robot is done inspecting all three location, it should back up and return home to its starting box to recharge.

Rules and Procedures:

- Create the robot’s starting area with electrical tape that is slightly larger than the robot.
- Use the electrical tape to mark three (3) inspection points along the robots path. The exact location are not important, but they should not be moved once the board is finalized.
- Robot must start inside the starting box (no parts over the line) and with its arm raised.
- The robot must move and stop at each line, lowering and raising its arm, representing the inspection process. The arm must be directly over each line when the inspection is performed.
- The robot must return to its starting box after completing the inspection process at the third line. The entire robot must be inside the box (no parts over the line).

Hints:

- Use the Robot Virtual World Measurement Tool Kit to measure the distances to each line on the board so you know how far you need to move each time!
- Try finding the number of centimeters your robot travels in each rotation, and using that to find the number of rotations you need
- You can also make a test run, then calculate “how many times as far” you need to move to get to each line, compared to the test run
CHAPTER 2: Orchard Robot Virtual World Challenge

In this challenge, you will program the robot to move from its starting area through three rows of fruit trees. You may choose your own path through the orchard, but the robot must pass alone both sides of each row during its run.

Rules and Procedures:

- For this challenge, the user can create their starting area wherever on the board.
- Use three strips of electrical tape to mark three rows of trees. The exact locations are not important, but they should not be moved once the board is finalized.
- Make sure there is enough space between the rows for the robot to pass on both sides of each row without crossing the lines.

Hints:

- Use the Robot Virtual World Measurement Tool Kit to measure the distances and angles so you know how far you need to move each time.
- Try finding the number of centimeters your robot travels or number of degrees its body turns in each wheel rotation.
- You can also make the test run, then calculate “how many times as far” you need to move or turn to get the amount of movement you want, compared to a test run.
CHAPTER 3: Arm Position Robot Virtual World Challenge

In this challenge, you will program the robot’s arm to move into the “Up” position when the “Up” button on the EV3 is pressed, no matter where the arm started. The robot will then move forward five (5) rotations to pick up a cargo container, and bring it back to the starting location.

Rules and Procedures:

- The robot’s arm will be moved to a random position before running.
- You must use parts available to you to modify the Touch Sensor’s mounting so that it can detect when the EV3RE’s arm is in the “up” position.
- Try to do this with as few changes as possible.
- When the “Up” button on the front the EV3 is pressed, the robot must raise its arm to the “Up” position (1), move forward two rotations to the cargo (2), and bring it back to the original starting position (3).

Hints:

- The trigger area on the Touch Sensor is small. You will probably need to build a “bumper” or “extender” on the end of th Touch Sensor to make it detect the arm more reliably.
- The EV3 core set includes two Touch Sensors. You can use one sensor for detecting the arm in the “Up” position, and the other to detect the box.
- You can also make a test run, then calculate “how many times as far” you need to move or turn to get the amount of movement you want, compared to a test run.
CHAPTER 4: Maze Challenge

In this challenge, you will program the EV3 robot to move from its starting area through a maze with tall, vertical walls. Walls in the maze can be adjusted to be nearer or farther between each run, but the general path must remain unchanged. Make use of the Ultrasonic Sensor and be sure to not touch any walls along the way.

Rules and Procedures:

- Recreate the maze with tall objects, such as books or binders.
- The robot must start entirely within the start box (1), as well as stopping entirely inside the end box (2).
- Walls marked by gray lines can move slightly.
- Walls marked by dark black lines cannot be moved.
- The robot must not touch any walls as it navigates throughout the maze.

Hints:

- The patterns of turns cannot change, so you don’t need to worry about using a sensor (other than Rotation) during turns.
- Look carefully at what distances in the maze are guaranteed not to change. Try to use those distances in combination with the Ultrasonic Sensor to orient yourself.
CHAPTER 5: Mower Robot Virtual World Challenge

In this challenge, you will program your EV3 robot to erase or clear the entire gameboard of either markings or parts. The robot is able to move freely in straight lines, using any method you want. However, there are three mud zones marked on the game board. When turning in one of these areas, the robot must be picked up by hand, and placed back down.

Rules and Procedures:

- The robot must clear the board of marks or parts to complete its mission.
- If any part of the robot is in the mud zone (red squares) at any point during a turn, it must be picked up and placed back down, as close to the same spot and facing as possible.
- Use overlapping paths to compensate for sensor inaccuracies

Hints:

- The Gyro Sensor responds only to the robot’s body turning, and is unaffected by interruptions like being picked up.
- It is unlikely that the eraser (or scoop) will perform perfectly, especially near the edges of its reach. Plan your robot’s course accordingly.
- Use an adjustment factor to compensate for the fact that the robot won’t see 90 degrees until it is past 90 degrees.
CHAPTER 6: Traffic Signal Challenge

In this challenge, you will program your EV3 robot to through three different intersection, each of which has a traffic signal. The traffic signal, which can be either the colored block or the red/green card, is held by hand at a set height. Unlike a camera, the detection range of the Color Sensor is short, so you will need to modify its placement on the robot so that it can see the traffic signal and react appropriately.

Rules and Procedures:

- Traffic signals are represented by holding the colored block with the signal side downward and toward the robot at a 45-degree angle.
- You can also use red and green colored sheets of paper, which may be easier to hold at the correct height.
- The initial signal color for each intersection is determined by flipping three coins before the run. Heads = Green, Tails = Red.
- The robot MUST stop if the light is red, and MUST NOT stop if the light is green.
- After the robot successfully gotten through all three intersections, the robot can be stopped by hand.
- BONUS: Write the program so that it DOES always stop after going through the third intersection. This will require the use of a Switch.

Hints:

- Since you only have access to Wait commands, try to break the robot's behavior into stages. What is the robot waiting to see at each stage of its movement? What should its motors be doing during those stages?
- If the robot is already moving, is there any point in waiting for Green? If the robot is stopped, is there any point in waiting for Red?
- If the robot sees a Red light, don’t forget that it needs to wait for the actual Green light before proceeding!
CHAPTER 7: Container Handling Robot Virtual World Challenge

In this Challenge, you will use a Loop to program your robot to move a series of containers into a loading zone. The containers to be loaded are placed at irregular intervals, so you will have to use a sensor to detect each one. The robot should then use its arm to transport the container back into the loading zone – marked with a red outline – and release it there.

Rules and Procedures:

- Use four rectangular objects to represent the four containers. These can be built with LEGO parts, or cut from cardboard boxes.
- The four objects must be placed in a straight line at the start of each run, with a flat side facing the robot, but their distance from each other should be varied randomly.
- The drop-off area for the containers should be marked with red electrical tape. If red is not available, any color may be substituted as long as it is different from the color of the table surface.
- A container with no parts hanging outside the loading area may be removed by hand to make room for the next container.
- The robot must return all four containers reliably to the loading area, regardless of where they were located along the initial line.

Hints:

- Use a Loop to repeat similar portions of the behavior. You should not have to write all four runs separately!
- If the robot needs to perform any actions prior to the loop, simply place them before the Loop in the program.
- If the robot needs to perform any actions after the loop, simply place them after the Loop in the program.
- Use Sensors to help the robot locate both the containers and the loading zone, as the exact distances required will be different each time the robot runs!
CHAPTER 8: Strawberry Plant Virtual World Challenge

In this Challenge, you will use a combination of both Loops and Switches to repeat an inspection process on a total of four plants, represented as boxes. The robot will move to a plant, then sort it to either the robot’s right side if it is good, or the robot’s left if it is bad.

Rules and Procedures:

- Use four rectangular objects to represent the plants. These can be built with LEGO parts, or cut from cardboard boxes.
- The objects must be placed in a straight line, with at least 5 centimeters’ space between them.
- The objects should be a random mix of “good” and “bad” plants, and placed in a random order for each run.
- Green or brown plants are considered “good”.
- Red, black, or yellow plants are considered “bad”.
- The robot must correctly sort all plants in a run to complete the Challenge.

Hints:

- Use a Switch to conduct individual inspections.
- Use a Loop to repeat the inspection process.
- According to the rules, when should the robot stop inspecting?
CHAPTER 9: Orchard Obstacle Virtual World Challenge

In this challenge, you will program your EV3 robot to move from its starting area through three rows of fruit trees. In addition, however, there will be one or more obstacles placed at random throughout the orchard. The robot should not touch these obstacles; instead, when it encounters one, it should stop moving until they are removed by hand... at which point the robot should continue on its way.

Rules and Procedures:

- This challenge uses the same game board layout as the Orchard Challenge from Chapter 2 (Turning).
- Like the previous challenge, the robot can start anywhere there is space available.
- Place one to two obstacles randomly alongside a side of a row for the robot to encounter.
- Be aware to not place an obstacle where the robot may bump into when turning a corner.
- When the robot encounters an obstacle, it should stop and wait for the Obstacle to be removed by hand. It should then continue moving without additional human intervention.

Hints:

- Use a meter stick of ruler to measure the distances to each line on the board so you know how far you need to move each time.
- The obstacle can be completely removed from the challenge after the robot approaches it and stops.
- Use lower speeds to minimize of effects of momentum when turning.
CHAPTER 10: Line Track Challenge

In this challenge, you will program your EV3 robot to grab a cargo crate from the pickup spot, follow the line track and drop the crate off in the drop-off zone.

Rules and Procedures:

- Build the course shown above using black electrical tape on a light surface.
- The robot must pick up the box using its arm, then follow the line to the drop-off point, and release the box there.
- Program your robot to deliver the box in the shortest time possible!

Hints:

- You can use the Loop Mode setting to adjust when a Line Track behavior ends. The program will then run whatever comes next, even another Line Track!
- Use multiple line tracks with different “sharpness” one after another to handle parts of the board.
- Adjust the “sharpness” of the robot’s motions using the Steering slider on the Move Steering Block.
- You may find it advantageous to track the right side of the line in some places. Which way should the robot go when it sees Black to get to the right edge of the line? Which way should it go to get back after it drives off?
Project Planning Engineering Journals

Keeping Track of the Project

An Engineering Journal is a place to keep track of all of the important information for the project. It can be a folder, binder or notebook, or it can be digital (Google docs) as long as it is wholly devoted to this project alone. In it, you will keep all of the notes, handouts, sketches and assignments that are related to the project.

Each person/team is responsible for keeping an Engineering Journal. It is important to keep the journal in order, updated, and safe. Your team will use the journal to keep ideas, document changes, and document all parts of the project.

All material should be kept in chronological order

Your personal Engineering Design Notebook will include:
• Class handouts
• Daily logs and notes
• All sketches, plans, and drawings
• Notes from design reviews
• Calculations relevant to your project
• Documentation of the evolutionary changes of your project
• All completed and returned assignments
• Final (turned-in) version of any individual assignments that are due

Your team’s Journal will include:
• Research information, such as computer print-outs and newspaper articles
• Meeting minutes or logs, including explanatory sketches and concept drawings
• Scheduling tools like PERT or Gantt Charts
• Notes for presentations, reports, proposals, etc.
• Final (turned-in) version of any group assignments that are due
• Graded and returned group assignments

Note: If your group only needs to keep one notebook for the whole group, choose a group member to be responsible for it so it does not get lost!

Assessment

The notebook itself will be graded based on completeness and organization

Students are responsible for lost, damaged, or poorly kept Journals. There will be no credit given for lost Engineering Journals

Notes and logs are the only evidence of work done on a daily basis. Make sure they are complete and fully explain your individual contributions to the project
Intro to the Engineering Journal

The Engineering Journal
All students are required to keep an Engineering Journal. It consists of a folder or binder where students store their classwork.

Each student’s Engineering Journal contains:
• All class handouts
• All student daily logs and class notes
• All completed and returned assignments
• Final (turned-in) version of any individual assignments that are due

All material should be kept in chronological order

The Engineering Journal is your tool to keep your work organized!

Assessment
The Journal will be graded based on completeness, organization, and content
A complete journal should include:
- All class handouts, including syllabus and assignment sheets
- All teacher-assigned work (homework, quizzes, etc.)
- Daily logs, one per day of independent work
- All major project deliverables (proposal, plans, copies of the program)
- Group meeting notes

All documents in the journal should be organized by date

Students are responsible for lost, damaged, or poorly kept Journals
Points may be deducted for journals that are:
- Lost (no credit for assignments that are lost!)
- Damaged or sloppy (unprofessional!)

When requested, students should hand in their journals
This is the preferred method for collecting work on days assignments are due
- Penalties apply for groups or individuals who are not prepared

Journal contents are graded and returned in the journal
Assignments are graded according to their own rubrics
- Quizzes and journal hand-ins can be done together for convenience

Notes and logs are a student’s evidence of work done on a daily basis
Self- and peer-reported student records are how work habits are tracked
- Teamwork
- Effective use of time
- Good planning and preparation
Using the Engineering Journal

How to use the Engineering Journal in your classroom.

The first page in every Engineering Journal should include the project plan. This may be in the form of a PERT or Gantt Chart.

The rest of the Engineering Journal should contain all current and completed material by a student for the current project, in chronological order.

Team documents (meeting notes, student written pieces, copies of student pseudocode, programs, etc.) are included in the journal.

The Engineering Journal is the preferred method of collecting and organizing student-produced material for the Engineering Projects.
The teacher will be checking the engineering journal periodically. They may collect it for review, or inspect it during class.

All individual and group work records should be incorporated into the journals.

Daily logs are especially important during the student-directed portions of the development, because they will be your primary evidence of progress.

All students will turn in their journals at the end of a project. The assignment or records should be incorporated into the journals.

For group deliverables, students may only need to have the group journal (or one member’s per team) turned in.

Journals will be returned to students after they are graded.

Students are responsible for keeping their journals in good order and in good condition. Appearances count in the real world, they count here as well.
**PERT Charts**

A PERT chart is a project management tool used to schedule, organize, and coordinate tasks within a project. PERT stands for Program Evaluation Review Technique, it was developed by the U.S. Navy in the 1950s to manage the Polaris submarine program.

Some project managers prefer a PERT chart because of its ability to represent events and milestones of a project in a graphical method. The PERT chart shows which parts of the project need to be completed in a sequential method and which parts can be worked on simultaneously. The PERT chart is often preferred over the Gantt chart because it is able to clearly illustrate task dependencies. Frequently, project managers use both techniques.

**Development of a PERT chart**

The first thing that the team must do is divide their overall project into small tasks. Each small task should be assigned a leader. Then the team must sequence the order of completion. Some tasks of the problem may be completed independent of others. These tasks can be worked on simultaneously. Some tasks may need to be completed before others can start. These things must be worked on sequentially. The team will look at the tasks and try set deadlines based on prior experience. They should break tasks into the smallest parts possible to accurately calculate time.

1. Discuss the overall problem
2. Break the problem into small tasks
3. Sequence the order of completion of tasks
4. Schedule the tasks
5. Assign responsibility for tasks to teams and individuals
6. Meet regularly to check progress of overall project
7. Help each other complete tasks in a timely manner

**Sample PERT Chart**

Arrows illustrate places where one task depends on another.
Gantt Charts

In 1917 Henry L. Gantt, an American engineer and social scientist, developed a production control tool that subsequently has been named the Gantt chart. Gantt charts are useful tools for planning and scheduling projects.

Gantt charts allow project managers to:
1. Assess how long a project should take
2. Lay out the order in which tasks need to be carried out
3. Help manage the dependencies between tasks
4. Determine the resources needed
5. Help monitor progress

The Gantt chart provides a graphical illustration of a schedule that helps to plan, coordinate, and track specific tasks in a project. Gantt charts may be simple versions created on graph paper or more complex using project management applications such as Microsoft Project or Excel.

<table>
<thead>
<tr>
<th>Time</th>
<th>Tasks</th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Task 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Task 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Task 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Task 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A Gantt chart is constructed with a horizontal axis representing the total time span of the project, broken down into increments (for example, days, weeks, or months) and a vertical axis representing the tasks that make up the project (for example, if the project is outfitting your computer with new software, the major tasks involved might be: conduct research, choose software, install software).

<table>
<thead>
<tr>
<th>Time</th>
<th>Morning</th>
<th>Lunch</th>
<th>Afternoon</th>
<th>Evening</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Computer 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Computer 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Computer 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Computer 4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legend

- In Use
- Idle
- Maintenance

Gantt charts are also used by supervisors and team leaders to schedule and track the use of resources. This data can help the project manager determine schedule the optimal use of a technology.
The Design Review

One tool that is used to provide feedback from interested stakeholders is called a design review. During the review, the team presents the problem, technical problems that have to be overcome, potential solutions, resources needed, scheduling conflicts, and due dates. The goal of a design review is to get feedback from a larger number of people with the hope of building a better solution.

Regular Team Design Reviews

Teams need to meet regularly during the early stages of a project, but once the work starts if regular meetings aren’t scheduled, plans can become confused. It is important for the project manager to schedule regular design reviews (every other day, once a week, it will depend on your team’s situation). Regularly scheduled meetings allow all members of the team to see how the overall project is moving along. Additionally, they allow the project manager to look at the team’s resources and reallocate them as needed. Documentation of progress (or problems) should be kept, specific individual goals should be set, and schedules should be modified to accommodate the changes.

Detailed Design Reviews

Detailed Design Reviews enable the team to get feedback on the overall concept that they plan to use to solve the design problem. Interested reviewers are identified from the community, the reviewers may include parents, teachers, or team sponsors. During the review the team will present the problem, how the team’s initial solutions, how they have improved on the original solutions, and then asks the reviewers to give feedback and offer suggestions. Depending on the complexity of the problem, it may be appropriate to give reviewers documents to review before they come to the design review so they are able to properly prepare to provide useful feedback.

At the Design Review, the team will present the overall problem and how they intend to solve the problem. The presenters may have working prototypes, concept maps and sketches, and examples of code that demonstrate possible solutions.

IT IS IMPORTANT THAT DESIGN TEAMS ARE READY FOR REVIEWERS

It is embarrassing to all concerned if team members haven’t prepared properly. In the engineering world, design reviews are often scheduled so that funders can make decisions about funding projects. The team should make a concerted effort to be properly prepared when they meet with reviewers. If appropriate, the team will want to include examples of the following documentation: a set sketches, a Gantt or PERT chart to show proper planning, models and prototypes, pictures of similar solutions that the team found during the research stage, and sample programming logic or control examples (sensors & logic).
Robot Maze Challenge

In the Robot Maze Challenge, teams will be given a blueprint of a maze with exact dimensions right before the challenge. The robot must navigate the maze without human intervention, you are able to use remote control, but cannot touch the robot. Add 50% to the team’s final score if the robot can complete the maze autonomously. Add 25% to the final score if the robot uses a combination of autonomous and remote control navigation to complete the maze.

Rules
1. Teams consist of 2-4 students.
2. Teams must build and design their own code.
3. Teams must use EV3 parts.
4. Each robot will compete in three rounds with the best time being considered the official time. Teams can use up to 30 minutes between runs to modify their robot or program between rounds.
5. A 50% bonus will be added to the overall score if the robot is completely autonomous.
6. A 25% bonus will be added to the overall score if the robot uses a combination remote control and autonomous navigation.
7. A blueprint of the maze listing the exact dimensions will be given to the teams one hour before the competition.
8. The robot will be not be awarded points in the Performance/Final Operation section of the scoring if at any time during the navigation of the course it is handled by a human.

Team scoring is based on:
- Use of sensors
- Quality of the code
- Final operation
- Proof of teamwork
- Proof of planning (Engineering notebook)

Project Evaluation

<table>
<thead>
<tr>
<th></th>
<th>Possible Points</th>
<th>Self Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Well thought out use of sensors</td>
<td>20pts</td>
<td></td>
</tr>
<tr>
<td>Quality of the code</td>
<td>20pts</td>
<td></td>
</tr>
<tr>
<td>Consistency</td>
<td>20pts</td>
<td></td>
</tr>
<tr>
<td>Engineering Journal</td>
<td>20pts</td>
<td></td>
</tr>
<tr>
<td>Teamwork</td>
<td>20pts</td>
<td></td>
</tr>
<tr>
<td>Total Possible Points</td>
<td>100pts</td>
<td></td>
</tr>
</tbody>
</table>

Self Evaluation

Teams are required to complete a self evaluation of their project. Use the following references when self assessing.

- **Quality of design** - is your robot stable, are the sensors well placed, does your robot behave consistently?
- **Quality of Code** - is your code documented with comments? Does it work? Is there anything unique?
- **Consistency** - Does your robot complete the maze every time?
- **Proof of planning** - Do you have pseudocode or a flowchart?
- **Proof of Teamwork** - Did one person do everything? Can you show how you broke up the team’s responsibilities?

Notes:
Product Prototype Design

In the Product Prototype design challenge teams design a simple prototype of a system of their own choice using EV3 parts. For full credit the teams must develop a working prototype, a complete set of building instructions designed to allow others to build and test the system, product marketing materials, and product packaging materials.

Rules
1. Teams consist of 2-4 students.
2. Each team’s prototype must be student designed
3. All team documents must be kept in an Engineering Journal
4. All parts are EV3 parts
5. The team must develop a set of step by step plans that enable the system to be built by others.
6. If applicable, students should design the packaging and marketing for the product.
7. The working prototype must work!

Grading will be based on:
- Rigor of the design
- Product safety considerations
- Performance/Operation
- Building instructions
- Engineering Journal
- Marketing materials
- Product packaging
- Team presentation

Project Evaluation

<table>
<thead>
<tr>
<th></th>
<th>Possible Points</th>
<th>Self Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rigor of the design</td>
<td>10pts</td>
<td></td>
</tr>
<tr>
<td>Product safety</td>
<td>10pts</td>
<td></td>
</tr>
<tr>
<td>considerations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prototype performance</td>
<td>10pts</td>
<td></td>
</tr>
<tr>
<td>Building Instructions</td>
<td>10pts</td>
<td></td>
</tr>
<tr>
<td>Engineering Journal</td>
<td>30pts</td>
<td></td>
</tr>
<tr>
<td>Marketing materials</td>
<td>10pts</td>
<td></td>
</tr>
<tr>
<td>Packaging materials</td>
<td>10pts</td>
<td></td>
</tr>
<tr>
<td>Team presentation</td>
<td>10pts</td>
<td></td>
</tr>
<tr>
<td>Total Possible Points</td>
<td>100pts</td>
<td></td>
</tr>
</tbody>
</table>

Notes:

Self Evaluation
Teams are required to complete a self evaluation of their project. Use the following values to assess what you will be graded on. When judging the Engineering Journal multiple the score by three. For example, if your Engineering Journal was very good, then you would multiple 3 * 8 and your score would be 24 for the Engineering Journal.

0 - Not attempted
2 - Incomplete
4 - Needs improvement
7 - Average
8 – Very Good
10 - Excellent

In the Product Prototype design challenge teams design a simple prototype of a system of their own choice using EV3 parts. For full credit the teams must develop a working prototype, a complete set of building instructions designed to allow others to build and test the system, product marketing materials, and product packaging materials.

Rules
1. Teams consist of 2-4 students.
2. Each team’s prototype must be student designed
3. All team documents must be kept in an Engineering Journal
4. All parts are EV3 parts
5. The team must develop a set of step by step plans that enable the system to be built by others.
6. If applicable, students should design the packaging and marketing for the product.
7. The working prototype must work!

Grading will be based on:
- Rigor of the design
- Product safety considerations
- Performance/Operation
- Building instructions
- Engineering Journal
- Marketing materials
- Product packaging
- Team presentation

Project Evaluation

<table>
<thead>
<tr>
<th></th>
<th>Possible Points</th>
<th>Self Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rigor of the design</td>
<td>10pts</td>
<td></td>
</tr>
<tr>
<td>Product safety</td>
<td>10pts</td>
<td></td>
</tr>
<tr>
<td>considerations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prototype performance</td>
<td>10pts</td>
<td></td>
</tr>
<tr>
<td>Building Instructions</td>
<td>10pts</td>
<td></td>
</tr>
<tr>
<td>Engineering Journal</td>
<td>30pts</td>
<td></td>
</tr>
<tr>
<td>Marketing materials</td>
<td>10pts</td>
<td></td>
</tr>
<tr>
<td>Packaging materials</td>
<td>10pts</td>
<td></td>
</tr>
<tr>
<td>Team presentation</td>
<td>10pts</td>
<td></td>
</tr>
<tr>
<td>Total Possible Points</td>
<td>100pts</td>
<td></td>
</tr>
</tbody>
</table>

Notes:

Self Evaluation
Teams are required to complete a self evaluation of their project. Use the following values to assess what you will be graded on. When judging the Engineering Journal multiple the score by three. For example, if your Engineering Journal was very good, then you would multiple 3 * 8 and your score would be 24 for the Engineering Journal.

0 - Not attempted
2 - Incomplete
4 - Needs improvement
7 - Average
8 – Very Good
10 - Excellent
# Student Work Habit Evaluation Rubric

<table>
<thead>
<tr>
<th>10 Advanced</th>
<th>9 Proficient</th>
<th>8 Basic</th>
<th>7-0 Unacceptable</th>
<th>Self</th>
<th>Teacher</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gives full attention to instructions and follows directions.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comes prepared and works the entire class period.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Works well with minimal supervision.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Works up to potential, shows maximum effort.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Works cooperatively as a member of a group.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Makes effective use of time and/or materials.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Demonstrates initiative and motivation.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Has a cooperative, positive attitude.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is on time for class.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Participates daily in the cleanup program.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Work Habits Point Total**
## Writing Rubric

<table>
<thead>
<tr>
<th></th>
<th><strong>Advanced</strong></th>
<th><strong>Proficient</strong></th>
<th><strong>Basic</strong></th>
<th><strong>Needs Improvement</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Content Understanding:</strong> The student explains the topic in an understandable way. (Does the student understand the topic?)</td>
<td>It is clear to the reader that the writer understands the topic and is able to support their understanding with facts and/or examples.</td>
<td>It is clear to the reader that the writer understands the topic at an advanced level.</td>
<td>The writer is able to communicate basic understanding of the topic</td>
<td>The writer does not provide relevant details and facts and it is not clear that they know the topic.</td>
</tr>
<tr>
<td><strong>Cohesion:</strong> The student response uses effective writing and communication techniques to convey meaning clearly. (Is the material easy to read?)</td>
<td>The response is well-written and conveys meaning clearly. It uses connective language efficiently to enhance meaning (e.g. but...while) and is easy to read.</td>
<td>The response is a readable and cogent response that answers the question clearly, but it includes minor grammatical errors that do not interfere with meaning.</td>
<td>The response includes list-type responses, or uses inappropriate connective language that detracts from the intended meaning and makes the response difficult to read.</td>
<td>The response is missing key elements, is incoherent, and it is difficult to read.</td>
</tr>
</tbody>
</table>
## Student Presentation Rubric

<table>
<thead>
<tr>
<th>Use of Multimedia Technology</th>
<th>Advanced</th>
<th>Proficient</th>
<th>Basic</th>
<th>Below Basic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Excellent use of multimedia technology.</td>
<td>Student demonstrated they knew how to use multimedia technology.</td>
<td>Multimedia technology use needs work.</td>
<td>Multimedia technology didn’t support topic.</td>
<td></td>
</tr>
<tr>
<td>• The presentation was eye appealing.</td>
<td>• The presentation was eye appealing.</td>
<td>• The use of multimedia technology was a distraction rather than a help.</td>
<td>• Pictures were not clear and didn’t seem to have a purpose.</td>
<td></td>
</tr>
<tr>
<td>• The pictures were clear.</td>
<td>• The pictures were clear.</td>
<td>• Incomplete</td>
<td>• Incomplete</td>
<td></td>
</tr>
<tr>
<td>• The sequence of the presentation was well thought out.</td>
<td>• The sequence of the presentation was well thought out.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Presentation was organized,</td>
<td>• Complete</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Speakers were clear and used proper terminology.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Complete</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Content Analysis</th>
<th>Advanced</th>
<th>Proficient</th>
<th>Basic</th>
<th>Below Basic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content was excellent and well supported with data and examples.</td>
<td>Content was good but could have been better supported with data and examples.</td>
<td>Content of presentation lacked clarity.</td>
<td>Content of presentation was incomplete and lacked clarity.</td>
<td></td>
</tr>
<tr>
<td>• The presentation was organized.</td>
<td>• The presentation was organized.</td>
<td>• The presentation lacked organization and didn’t have a unified theme.</td>
<td>• The presentation was incomplete.</td>
<td></td>
</tr>
<tr>
<td>• The project was fully described.</td>
<td>• The presentation was organized but needed more practice to be excellent.</td>
<td>• The presenters didn’t use proper terminology.</td>
<td>• The presenters were not ready to present.</td>
<td></td>
</tr>
<tr>
<td>• The presentation included advanced relevant topics.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• The presentation was enjoyable to watch.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Design Review Rubric

<table>
<thead>
<tr>
<th></th>
<th>Advanced</th>
<th>Proficient</th>
<th>Basic</th>
<th>Below Basic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Timeliness (10%)</strong></td>
<td>• Design Candidate Sheets are completed on time for each design</td>
<td>• Design Candidate Sheets are complete but not on time</td>
<td>• Most Design Candidate Sheets are complete</td>
<td>• Design Candidate Sheets are not completed</td>
</tr>
<tr>
<td></td>
<td>• Design Assessment Criteria sheet is completed on time</td>
<td>• Design Assessment Criteria sheet is completed but not on time</td>
<td>• Design Assessment Criteria sheet is mostly complete</td>
<td>• Design Assessment Criteria sheet is not completed</td>
</tr>
<tr>
<td></td>
<td>• Group is present and ready to begin on time</td>
<td>• Group is present and ready to begin on time</td>
<td>• Group is present</td>
<td>• Group is not present</td>
</tr>
<tr>
<td><strong>Discussion (30%)</strong></td>
<td>• Group follows good meeting and teamwork procedures</td>
<td>• Group follows decent meeting and teamwork procedures</td>
<td>• Group follows few meeting and teamwork procedures</td>
<td>• Group does not work as a team</td>
</tr>
<tr>
<td></td>
<td>• Discussion remains professional in tone and direction</td>
<td>• Discussion remains professional in tone and direction</td>
<td>• Discussion does not have a professional tone or manner</td>
<td>• Little discussion occurs</td>
</tr>
<tr>
<td></td>
<td>• Discussion proceeds efficiently</td>
<td>• For the most part discussion proceeds efficiently</td>
<td>• Discussion does not proceed efficiently</td>
<td>• Discussion does not stay on topic</td>
</tr>
<tr>
<td></td>
<td>• Group is able to focus on the relevant aspects of the robot designs</td>
<td>• For the most part discussion focuses on relevant aspect of robot designs</td>
<td>• Group rarely focuses on relevant aspects of robot designs</td>
<td>• Group does not focus on relevant aspects of robot design</td>
</tr>
<tr>
<td><strong>Problem (40%)</strong></td>
<td>• Design Assessment Criteria are appropriate</td>
<td>• Design Assessment Criteria are mostly appropriate</td>
<td>• Design Assessment Criteria are not very appropriate</td>
<td>• Design Assessment Criteria do not exist</td>
</tr>
<tr>
<td></td>
<td>• Discussion indicates that all team members are familiar with the problem</td>
<td>• Discussion indicates that most team members are familiar with the problem</td>
<td>• Discussion indicates that a few team members are familiar with the problem</td>
<td>• Little discussion takes place</td>
</tr>
<tr>
<td></td>
<td>• Discussion indicates that all team members understand the needs of the solution</td>
<td>• Discussion indicates that most team members understand the needs of the solution</td>
<td>• Discussion indicates that a few team members understand the needs of the solution</td>
<td>• Candidate designs do not exist</td>
</tr>
<tr>
<td></td>
<td>• Candidate designs are oriented toward solving the problem</td>
<td>• Candidate designs are mostly oriented towards solving the problem</td>
<td>• Candidate designs do not really try to solve the problem</td>
<td>• Candidate designs do not exist</td>
</tr>
<tr>
<td></td>
<td>• Candidate designs show evidence of thought out design including mechanics, programming, and testing</td>
<td>• Candidate designs mostly show evidence of thought out designs</td>
<td>• Candidate designs do not show a thought out design</td>
<td></td>
</tr>
<tr>
<td><strong>Consensus (20%)</strong></td>
<td>• Group members avoid unnecessary “attachment” to their designs that gets in the way of productive discussion</td>
<td>• Group members avoid unnecessary “attachment” to their designs that gets in the way of productive discussion</td>
<td>• Group member must keep pieces of their original design which may temporarily halt productive discussion</td>
<td>• Group members feel that their design is the only design</td>
</tr>
<tr>
<td></td>
<td>• All group member are able to reach consensus</td>
<td>• Most group members are able to reach a consensus</td>
<td>• Few group members reach consensus</td>
<td>• Group members never reach consensus</td>
</tr>
</tbody>
</table>
## Proposal Writing Rubrics

<table>
<thead>
<tr>
<th></th>
<th>Advanced</th>
<th>Proficient</th>
<th>Basic</th>
<th>Below Basic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Timeliness</strong> (10%)</td>
<td>• All required elements are produced on time</td>
<td>• Most required elements are produced on time</td>
<td>• Few required elements are produced on time</td>
<td>• Required elements are not turned in</td>
</tr>
<tr>
<td><strong>Presentation</strong> (20%)</td>
<td>• Proposal is well written with no grammatical or spelling errors</td>
<td>• Proposal is well written with few grammatical or spelling errors</td>
<td>• Proposal is fairly well written with many grammatical or spelling errors</td>
<td>• Proposal is not well written and has many grammatical or spelling errors</td>
</tr>
<tr>
<td></td>
<td>• Proposal has been reviewed at least once</td>
<td>• Proposal has been reviewed at least once</td>
<td>• Proposal has never been reviewed</td>
<td>• Proposal has never been reviewed</td>
</tr>
<tr>
<td></td>
<td>• All required elements are included</td>
<td>• Most required elements are included</td>
<td>• Most required elements are included</td>
<td>• Few required elements are included</td>
</tr>
<tr>
<td></td>
<td>• Time lines and charts are written clearly, with no unnecessary marks or cross-outs</td>
<td>• Time lines and charts are written clearly with few unnecessary marks or cross-out</td>
<td>• Time lines and charts are written clearly with many unnecessary marks and cross-outs</td>
<td>• Time lines and charts are not written clearly with many unnecessary marks and cross-outs</td>
</tr>
<tr>
<td><strong>Practicality</strong> (25%)</td>
<td>• Proposed solution demonstrates understanding of real-world constraints (i.e. laws of physics, time)</td>
<td>• Proposed solution demonstrates a fair understanding of real-world constraints (i.e. laws of physics, time)</td>
<td>• Proposed solution demonstrates a poor understanding of real-world constraints (i.e. laws of physics, time)</td>
<td>• No proposed solution is given</td>
</tr>
<tr>
<td></td>
<td>• Time line specifies due dates for required deliverables</td>
<td>• Time line specifies most due dates for required deliverables</td>
<td>• Time line specifies few due dates for required deliverables</td>
<td>• No time line indicated</td>
</tr>
<tr>
<td></td>
<td>• Materials list is reasonable, given resources</td>
<td>• Materials list is mostly reasonable, given resources</td>
<td>• Materials list is not reasonable, given resources</td>
<td>• Materials list is not reasonable</td>
</tr>
<tr>
<td></td>
<td>• Proposal clearly links problem to proposed solution</td>
<td>• The proposal mostly links problem to the proposed solution</td>
<td>• Very little connection made between the proposed solution and the problem</td>
<td>• No connection made between the proposed solution and the problem</td>
</tr>
<tr>
<td><strong>Problem Understanding</strong> (25%)</td>
<td>• Proposal demonstrates clear understanding of problem</td>
<td>• Proposal shows a good deal of understanding of problem</td>
<td>• Proposal shows little understanding of problem</td>
<td>• Proposal demonstrates no understanding of problem</td>
</tr>
<tr>
<td></td>
<td>• Shows consideration for need and potential users of product</td>
<td>• Shows a good deal of consideration for need and potential of users of product</td>
<td>• Shows little consideration for need and potential of users of product</td>
<td>• Shows no consideration for need and potential users of product</td>
</tr>
<tr>
<td><strong>Teamwork</strong> (10%)</td>
<td>• Team has defined appropriate roles/responsibilities for all members</td>
<td>• Most of the team has defined roles/responsibilities</td>
<td>• Few members of the team have defined roles/responsibilities</td>
<td>• No roles/responsibilities were defined for group members</td>
</tr>
</tbody>
</table>
Robot Physics Terms

In your robotic labs there are many physics concepts demonstrated. Your team’s job is to develop a short presentation that includes five or more of the physics terms below and present it to the class. Your team is encouraged to use references from the Internet, physics texts, or your science book. For example, the definitions below were found by typing in “define: keyword” in Google. Example: define: potential energy.

Potential energy - The energy stored in a raised object (e.g., the weights in a grandfather clock). Potential energy equals $mgh$, where $m$ is mass, $g$ is the acceleration of gravity, and $h$ is the vertical distance from a reference location. It is called potential energy because the energy can be regained when the object is lowered. This type of potential energy is sometimes called gravitational potential energy in order to distinguish it from elastic potential energy:

Elastic potential energy - The energy stored in deformed elastic material (e.g., a watch spring). Elastic energy equals $\frac{1}{2}kx^2$, where $k$ is the stiffness, and $x$ is the associated deflection. Elastic energy is sometimes called elastic potential energy because it can be recovered when the object returns to its original shape.

Mass - the property of a body that causes it to have weight in a gravitational field.
– Mass is a property of physical objects that, roughly speaking, measures the amount of matter they contain. It is a central concept of classical mechanics and related subjects.

Gravity - the force of attraction between all masses in the universe; especially the attraction of the earth’s mass for bodies near its surface; “the more remote the body the less the gravity”; “the gravitation between two bodies is proportional to the product of their masses and inversely proportional to the square of the distance between them.

Speed - Speed (symbol: $v$) is the rate of motion, or equivalently the rate of change of position, expressed as distance $d$ moved per unit of time $t$. i.e. Meters per second, kilometers per hour, miles per hour,

Motion - a natural event that involves a change in the position or location of something

Kinetic energy - Energy that a body has as a result of its motion. Mathematically, it is defined as one-half the product of a body’s mass and the square of its speed.

Momentum - The product of the mass times the velocity of an object.

Inertia - The tendency of an object at rest to remain at rest, and of an object in motion to remain in motion.

Friction - the resistance encountered when one body is moved in contact with another.

Kinetic Friction - The frictional force tending to slow a body in motion.

Static Friction - Friction at rest; a force is required to initiate relative movement between two bodies - static friction is the force that resists such relative movement. Sometimes referred to as stiction.

Mechanical advantage - The ratio of output force to input force for a machine.

Conservation of energy - a law of physics that states that energy can not be created or destroyed only converted from one form to another.
The process of engineering begins with an idea. The ability to communicate many technical ideas requires a translation from thoughts into pictures. A quick way to share and idea is through a technical sketch. Technical sketching is a tool used by engineers and inventors daily. If the idea turns out to be a good one, these first sketches are turned into a full set of drawings, or solid models, which include measurements and other critical details. From the initial sketches, the first prototypes can be developed. The freehand sketch is the first step taken to turn an idea into reality.

**Developing Proper Sketching Technique**

All two dimensional pictures can be broken into straight lines and curved lines. The next couple of exercises will help the novice to develop good sketching technique. We will start by practicing straight, parallel lines.

Things to remember:
1. Hold your pencil using a loose grip.
2. Lock your wrist and move your whole shoulder as you sketch you straight lines.
3. Guide the pencil with your eye by continually looking toward where you would like the pencil to go.
4. Keep your lines light and consistent.

**Straight lines**

**Sketch exercise one**

Begin by selecting a sharp pencil and a blank piece of paper. In this sketching exercise you will sketch straight light lines keeping them evenly spaced all the way down the paper. Initially, you may want to place a couple of light points that you can look towards to guide your pencil. Soon you will be sketching straight light lines easily. Remember to look toward where you would like the pencil to move. Keep all lines light. Hold the pencil loosely.

**Sketching a Square**

1. Sketch the first side of your box, keep the line parallel with the top of the paper.
2. Sketch the second side of the square. Keep the line parallel with the first line.
3. Sketch the third side. Keep the line perpendicular to the first two lines.
4. Sketch the last line. Make sure that the last line is sketched placed in a position that makes both sides equal.

**Sketch exercise two**

Begin with a scrap, blank sheet of paper. Sketch a small square in the middle of the paper. Sketch another square around the first square. Maintain an equal distance between squares.
Technical Sketching 2 of 2

You do not have to be an artist to accurately and neatly describe your ideas using technical sketching. A good example of layout is demonstrated at the right - laying out a circle. A circle fits perfectly into a square. In order to draw proportionally correct circles use the following sequence:

1. Begin by sketching a square.
2. Proportionally divide the square into four even areas.
3.- 6. Complete the four semi-circles.

Note: Remember to keep all of your lines light. Darken when the sketch is correct.

Sketch exercise three

This sketching exercise is designed to give the beginner practice sketching straight and curved lines while keeping things proportional. Start by sketching a square in the middle of your paper then continue to add squares and circles until the paper is filled.

Additional Sketching Exercises

Use blank sheets of paper to sketch the shapes shown in the boxes below. You will need a sharp pencil. For extra independent practice complete the example sketches below.

Things to remember:

1. Don’t grip the pencil too tightly.
2. Keep developmental lines light.
3. Darken object lines when your drawing is complete.
4. Maintain proper proportion.
5. Keep your drawings neat.
Engineering Process

Research
- Identify problem
- Research current solutions
- Understand requirements
- Brainstorm solutions
- Develop concept prototypes

Choose an Idea

Plan
- Outline strategy
- Identify specifications
- Develop schedule

Submit a Proposal

Prototype
- Build working prototype
- Conduct design reviews

Revise based on Feedback

Test
- Iteratively improve
- Demonstrate Final Prototype

Commercialize
- Document & publish results
- Market
- Solicit customer feedback

Innovate Existing Solution

Improve based on customer feedback
Engineering Process

- **Research**
  - Choose an Idea

- **Plan**
  - Submit a Proposal

- **Prototype**
  - Revise based on Feedback

- **Test**
  - Demonstrate Final Prototype

- **Commercialize**
  - Innovate Existing Solution

*Improve based on customer feedback*