Learning with TBot

Version .3

Warranty

OneRobot warrants its products against defects in materials and workmanship for a period of 90 days from receipt of product. If you discover a defect, OneRobot will, at its option, repair or replace the merchandise, or refund the purchase price. Before returning the product to OneRobot, call for a Return Merchandise Authorization (RMA) number. Write the RMA number on the outside of the box used to return the merchandise to OneRobot. Please enclose the following along with the returned merchandise: your name, telephone number, shipping address, and a description of the problem. OneRobot will return your product or its replacement using the same shipping method used to ship the product to OneRobot.

14-Day Money Back Guarantee

If, within 14 days of having received your product, you find that it does not suit your needs, you may return it for a full refund. OneRobot will refund the purchase price of the product, excluding shipping/handling costs. This guarantee is void if the product has been altered or damaged. See the Warranty section above for instructions on returning a product to OneRobot.

Copyrights And Trademarks

Disclaimer Of Liability

OneRobot is not responsible for special, incidental, or consequential damages resulting from any breach of warranty, or under any legal theory, including lost profits, downtime, goodwill, damage to or replacement of equipment or property, or any costs of recovering, reprogramming, or reproducing any data stored in or used with OneRobot products. OneRobot is also not responsible for any personal damage, including that to life and health, resulting from use of any of our products. You take full responsibility for your OneRobot application, no matter how lifethreatening it may be.

Errata

While great effort is made to assure the accuracy of our texts, errors may still exist. Occasionally an errata sheet with a list of known errors and corrections for a given text will be posted on the related product page at www.OneRobot.org. If you find an error, please send an email to support@OneRobot.org.

Table of Contents

Preface

The Wonderful World of Robots

The TBot is a multipurpose educational robot that was designed to make learning both fun and exciting. It is suitable for a wide range of experiments and activities right from kindergartens up to university students. It has many features like motors control, full-colour LED lights, powerful sensors, encoders, sound playing, synthesizing, recording, plus many other features. Most interesting is that it can be programmed with the simple drag and drop 12Blocks language making it accessible to a wider range of users.

TBot is intended to help roboticists of various skill levels take their designs to the next level with microcontrollers and the knowhow to implement them effectively.

The goal of this text is to get students interested in and excited about the fields of engineering, mechatronics, and software development as they design, construct, and program an autonomous robot. This series of hands-on activities and projects will introduce students to basic robotic concepts using the OneRobot TBot.

The activities and projects in this text begin with setting up your programming environment to help you write your first TBot program and then move on to various activies that highlight different capabilities of the TBot.

Audience

This text is designed to be an entry point to technology literacy, and an easy learning curve for embedded programming and introductory robotics. The text is organized so that it can be used by the widest possible variety of students as well as independent learners. Middle-school students can try the examples in this text in a guided tour fashion by simply following the check-marked instructions with instructor supervision. At the other end of the spectrum, pre-engineering students' comprehension and problem-solving skills can be tested with the questions, exercises and projects (with solutions) in each chapter summary. The independent learner can work at his or her own pace, and obtain assistance through the forum cited below.

Support Forums

OneRobot maintains free, moderated forums for our customers, covering a variety of subjects at http://forums.hannoware.com including the 12Blocks programming language and the TBot robot.

Resources For Educators

To supplement our products, we provide a curriculum for the classroom. Designed to engage students, each lab contains full source code, online video, “How it Works” explanations, schematics, and wiring diagrams or photos for a device a student might like to use. Curriulum targetting different age groups are available online at http://onerobot.org/education.html

Educators Courses

These hands-on, intensive 2 hour virtual courses for instructors are taught by OneRobot engineers to prepare teachers for the classroom. Contact us for details at http://onerobot.com/contact.html

Copyright Permissions for Educational Use

About The Authors

Chad George earned a degree in Computer Science from Indiana University where he coauthored two papers on biologically inspired artificial intelligence. Working in the manufacturing industry he has engineered a number of control systems requiring the design of custom PCBs and embedded microcontrollers. He has also been very involved in STEM education, running summer camps, starting an after school STEM club and coaching FIRST Robotics teams. Seeing a need for a low cost, highly capable robotics system for educational environment, he designed the TBot robot platform. Chad continues to work on improving the TBot and other hardware projects that support the OneRobot goals of improving STEM education through hands on access to robotics.

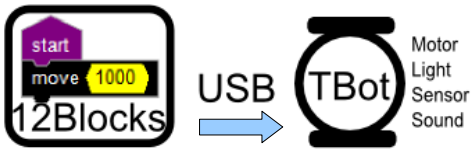
Hanno Sander earned a degree in Computer Science from Stanford University, where he built one of the first hybrid cars, collaborated on a microsatellite, and studied artificial intelligence. He later founded a startup to develop customized information services and then transitioned to product marketing in Silicon Valley with Oracle, Yahoo, and Verity. Today, Hanno’s company HannoWare seeks to make sophisticated technology—robots, programming languages, debugging tools, and oscilloscopes—more accessible. Hanno lives in Christchurch, New Zealand, where he enjoys his growing family and focuses on his passion of improving education with technology.

Special Contributors: Ingolf Sander, Professor Tim Bell, Steve Woodrough

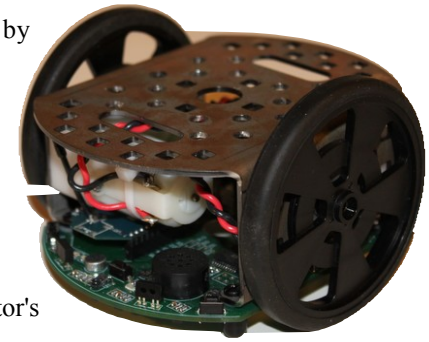
Chapter 1: Getting Started

By the end of this chapter you'll be writing your first program for your TBot! Let's get started with an overview of what's involved.

Two powerful microcontrollers control your Tbot's motors, lights, sounds, communication and sensors. To control your TBot you'll write programs using the 12Blocks visual language and download them using a USB cable. The following figure illustrates these concepts:



Step 1: Connect TBot to your PC with a USB cable



To program your TBot you need to connect it to your PC with a USB cable. Start by connecting the mini-USB male connector to the rear of your TBot. Finish by connecting the USB cable to your PC. You do not need a PropPlug or other USB to serial adapter.

Your TBot batteries will automatically be charged when connected to USB. Just like other USB devices- for example, the Apple iPod- you don't have to worry about its state of charge- the TBot charges itself.

When connected to USB your TBot will run from your PC's power to allow programming, play sounds, and manipulate it's LED lights. To run the TBot's motor's for movement, you'll need to slide the power switch to “ON”.

When your TBot is not connected to USB, it will run the last program stored into it's permanent memory (EEPROM) when you slide the power switch to “ON”.

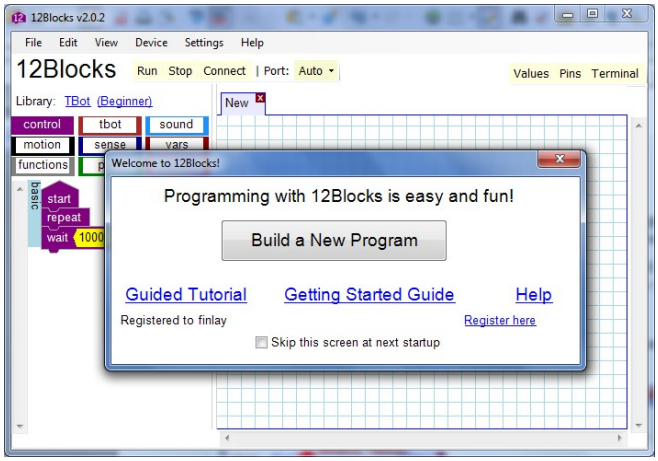
Step 2: Install Software

12Blocks is a visual and easy to use drop-and-drag programming language that is highly suitable for educational purposes. It is mainly targeted at programming robots and microcontrollers and supports various types of devices.

System Requirements: You will need a personal computer to run the 12Blocks software. Your computer will need to have the following features:

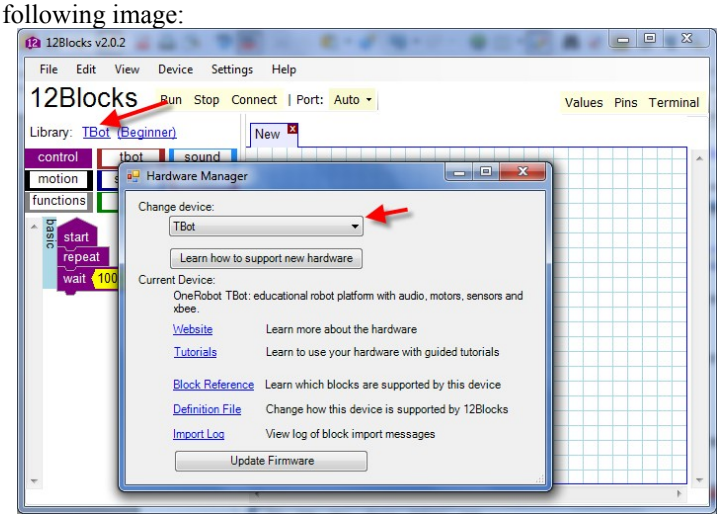
- Microsoft Windows 2K/XP/Vista/7 or newer operating system (Beta for OSX and Linux is available)
- An available USB port
- Internet access and an Internet browser program

Download the 12Blocks installer from http://12blocks.com. A wizard interface will help you install and configure the program for your PC. The program will start automatically and show the following introductory screen:



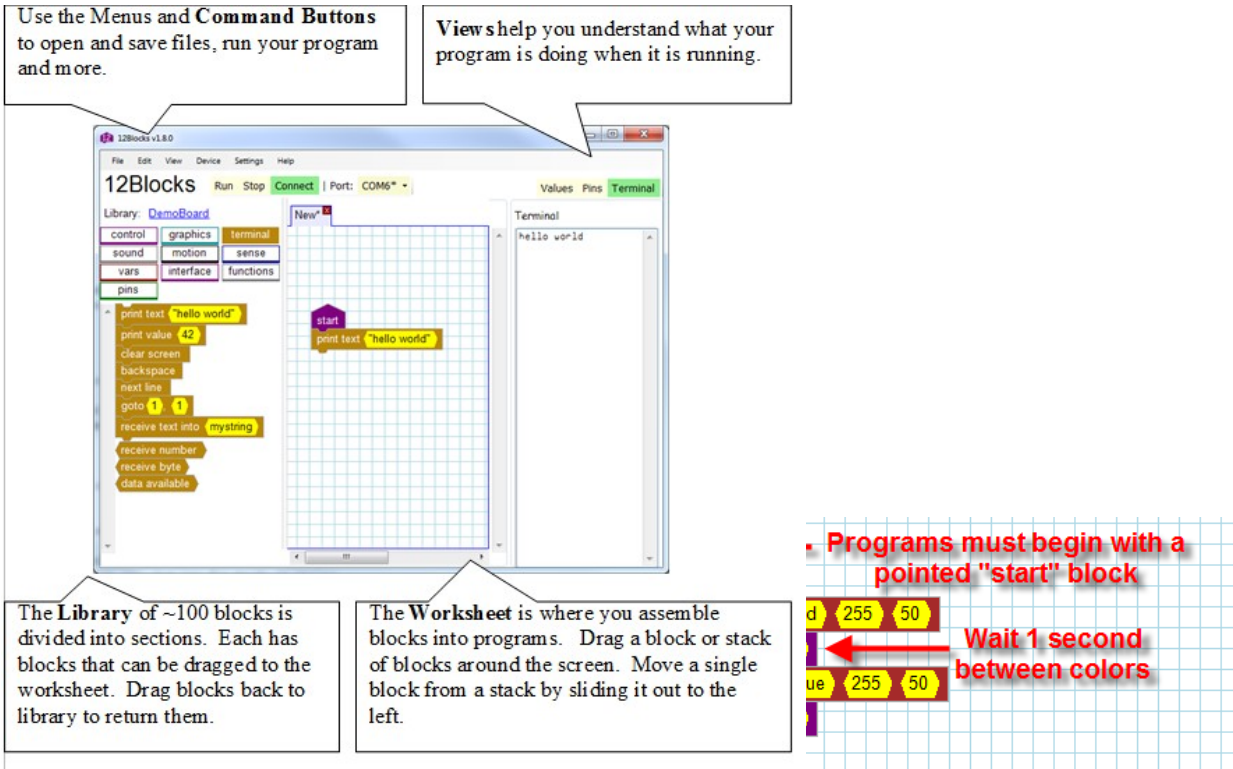
You'll need to follow the “Register here” link to register your software with the license key provided to you when you purchased your TBot.

12Blocks supports a range of popular microcontrollers and robots- for use with your TBot you must select the TBot library as indicated in the following image:



Close the Hardware Manager to finish setting up your 12Blocks environment- you are now ready to get started with robotics!

Step 3: Your First Program



For our first program we'll blink the Tbot's full color LED by setting it to repeatedly changing it's color. Drag the blocks to the worksheet and press “run”.

Concepts:

- Use a “start” block to begin a stack of blocks
- A “repeat” block will repeat the blocks attached inside of it
- Use the “set LED” block from the “TBot” section to set the color, saturation and brightness of the light emitting diode. Click on the yellow part of a block to change a parameter.
- The “wait” block pauses the program for a number of milliseconds. If you don't use it, the color changes will happen too quickly for you to notice.

Hints and Tips

- Press F1 to access Help for a reference of all blocks, language comparison, etc.
- 'Guided Tutorials' on the “File” menu show you how to program visually
- 'Example files' on the “File” menu gives you samples to learn from
- Store a program permanently on the TBot by pressing “Device>Load Permanently”
- Your TBot will charge when connected via USB
- To use the motors while connected via USB switch your TBot to “ON”
- Connect your TBot to other sensors and devices with the expansion connecter- refer to Chapter 6 for details.

Chapter 2: Simple Programs

Basic Movement

Now it's time to make the robot move. Make sure the TBot is in a safe position, then switch it to “Run with motors” before loading it.

Concepts:

- “File>New” to start a new program
- “File>Save” to save it
- “move” and “turn” are in the “motion” section
- Use negative numbers to turn left or move backwards

Navigation

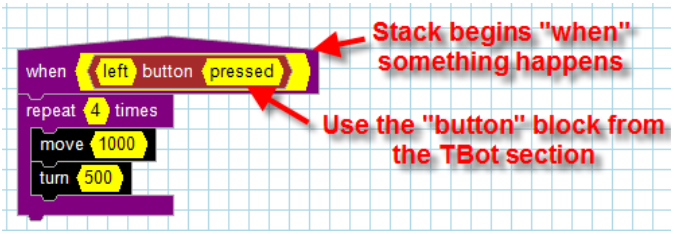
The TBot can be programmed to perform a variety of maneuvers. The maneuvers and programming techniques introduced in this chapter will be reused in later chapters. The only difference is that in this chapter, the TBot will blindly perform the maneuvers.

In later chapters, the TBot will perform similar maneuvers in response to conditions it detects with its sensors.

This chapter also introduces ways to tune and calibrate the TBot’s navigation. Included are techniques to straighten a TBot’s forward drive, more precise turns, and calculating distances.

Square Dancing

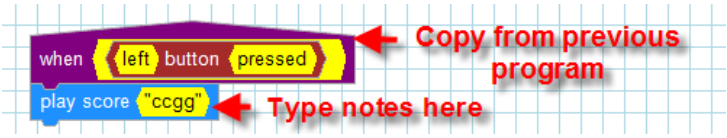
In the last program the TBot started moving right after we loaded it. Let's improve that so it moves after we push the left button. Also, use the “repeat x” block to repeat a set of blocks a number of times.



Music

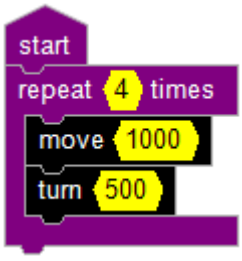
You can copy and paste blocks from one program to another by selecting the blocks you need and using the “Edit>Copy” menu

The “sound” section contains different blocks to make sounds. The “play score” blocks lets you enter notes. Make sure to use quotes.

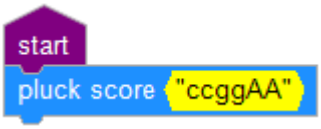


More Sample Programs:

1. Move in a square

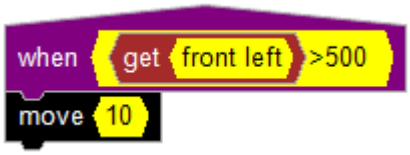


2. Play musical notes

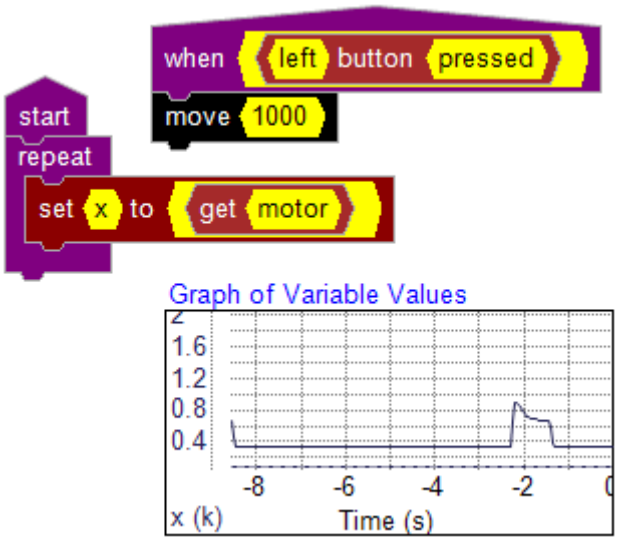


3. Flash colors when button is pressed

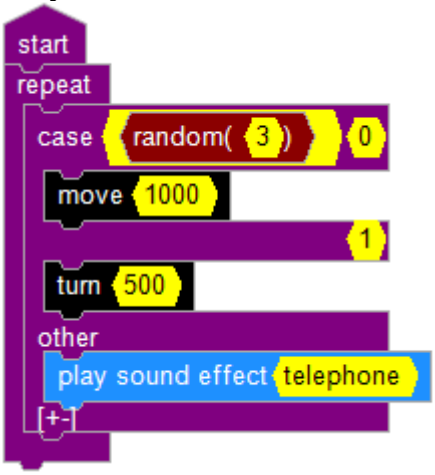
4. Move when front is clear



5. Move and graph motor current versus time



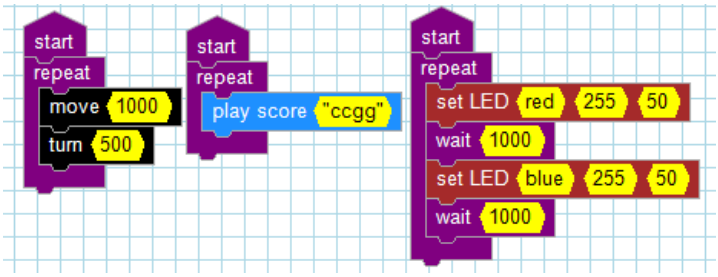
6. Randomly move and make sounds



Chapter 3: Advanced Programs

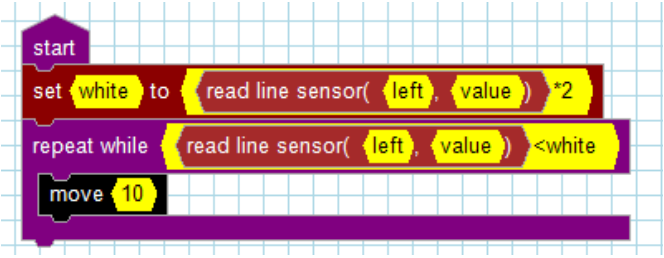
Multiprocessing

Use multiple “start” blocks to do multiple things at the same time.



Using Sensors to build a Shade seeker

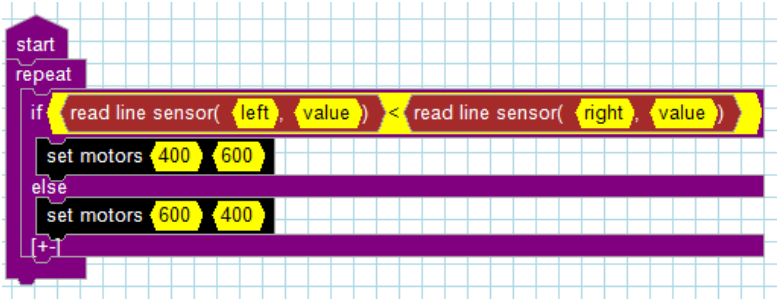
Now it gets a bit more complex! This program uses the TBot's line sensor to stop the TBot on a dark area. The sensor tells the TBot how dark the current surface is- larger values are darker. This program begins by taking a sample of the current area (assumed to be white) and multiplies it by 2- to ensure that slightly darker regions are recognized as white as well. It then repeats a small movement while additional readings are less than the value we recognize as white.



Line Follower

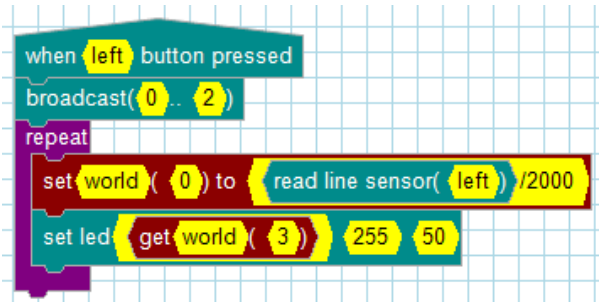
This program uses two line sensors to follow line. The “if” block uses logic to do one thing or another. “If” the parameter is “true” then the first block happens, otherwise the “else” part happens. We compare the left to the right line sensors and control the motor speed so the TBot tracks the line

a

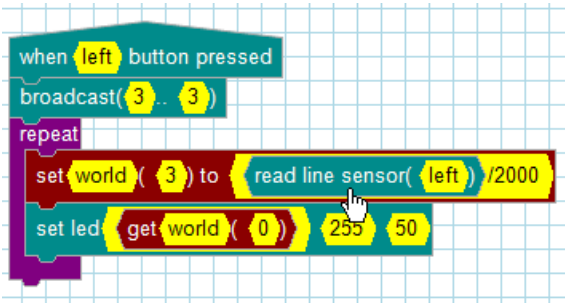


2 Robots Communicating via a shared “World” view

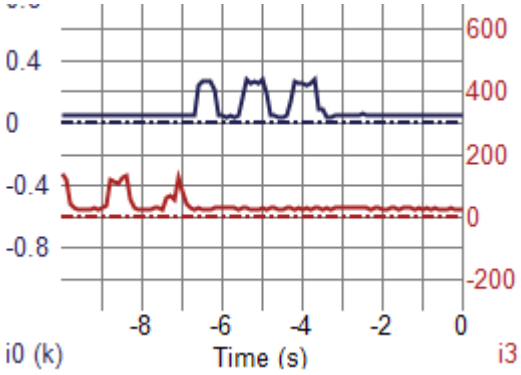
Bot #1 waits until the “left” button is pressed and then starts broadcasting elements 0..2 from the “world” array to all other bots. This “world” array is shared between all bots and 12blocks- and each element is available for graphing from within 12blocks. Bots use this “shared” memory to know the state of the world. They write to elements that they broadcast, and read from the other elements. This program sets world[0] to the line sensor measurement. You can graph this value, or use it to control other bots, just by reading from world[0]. The next line sets the led's hue to the value in world[3]- from below- you'll see that this is set to the line sensor reading of the other bot. So, this bot's led will change based on the position of the other bot.



Bot #2 sets world[3] to it's line sensor measurement, and sets the hue of the led to world[0]. It broadcasts world[3].



When moving both bots, 12block shows this reading:



Chapter 4: Sample Classwork

Lab 1 - Introduction to TBot

- Learn to use tool chain to write programs and load into robot
- Use user buttons and DIP switch to control program execution
- Read Sensors (Proximity, Microphone)
- Drive motors under basic sensor control
- Use LED/Speaker for representing robot status
- Begin to learn SPIN programming: basic syntax, flow control and program structure
- Learn to how to utilize premade SPIN objects to perform multiple tasks in parallel

Lab 2 - Line Follower

- Learn to use real-time debugging tools (Terminal, Data Signal Oscilloscope and Logic Analyzer)
- Read Line Sensors
- Analyze digital input/output signals controlling the line sensors and motors
- Control motors using sensors and reactive control loops
- Create custom SPIN object and process sensor data in parallel with motor control

Lab 3 - IR Remote Control

- Learn to program in Propeller Assembly
- Learn debugging techniques for assembly language programming
- Write a low-level driver to processes signals from an Universal IR Remote Control
- Control assembly language driver from high-level SPIN code to drive robot via remote control

Lab 4 - Maze Solver

- Learn to utilize multiple control techniques to complete a single task: reactive control, behavior
- based control and state machines
- Extend line following routines to handle corners and intersections
- Use proximity detectors to find "dead-ends"
- Create an internal model of the environment to solve problems

Chapter 5: Challenges and Competitions

Some of the following competitons are provided courtesy of Seattle Robotics Society.

Robot Floor Exercise

Purpose

The floor exercise competition is intended to give robot inventors an opportunity to show off their robots or other technical contraptions.

Rules

The rules for this competition are quite simple. A 10-foot-by-10-foot flat area is identified, preferably with some physical boundary. Each contestant will be given a maximum of five minutes in this area to show off what their robot can do. The robot's contestant can talk through the various capabilities and features of the robot. As always, any robot that could damage the area or pose a danger to the public will not be allowed. Robots need not be autonomous, but it is encouraged. Judging will be determined by the audience, either indicated by clapping (the loudest determined by the judge), or some other voting mechanism.

Line Following

Objective

To build an autonomous robot that begins in Area "A" (at position "S"), travels to Area "B" (completely via the line), then travels to the Area "C" (completely via the line), then returns to the Area "A" (at position "F"). The robot that does this in the least amount of time (including bonuses) wins. The robot must enter areas "B" and "C" to qualify. The exact layout of the course will not be known until contest day, but it will have the three areas previously described.

Skills Tested

The ability to recognize a navigational aid (the line) and use it to reach the goal.

Capture the Flag

Overview

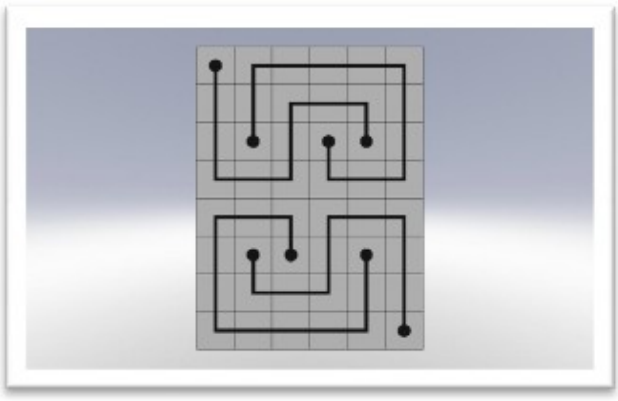
- Coordinate between multiple robots to complete a task
- Develop a protocol to handle communication between robots
- Create a strategy that maximizes the team's final score while following game rules

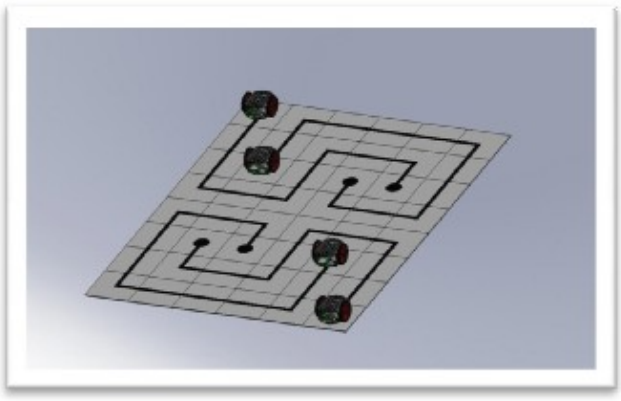
This is a competitive and cooperative challenge involving 2 teams of 2 robots per team. The objective of the game is follow a winding "maze-like" line from one end to the other in order to score points. Each robot will be equipped with a special beacon that it can activate when it reaches one end of the line. This active beacon becomes the "flag" that the robot must carry all the way to the opposite end in order to score points.

When a robot's beacon is activated it flashes a visible light, but it also begins transmitting an IR beacon for other robots to track onto using their front proximity sensors in passive mode. Each beacon unit is also equipped with several IR remote receivers. Opposing robots can "tag" the robot trying to make a flag run by sending a well aimed IR signal at the active beacon. A successful tag will prevent the team from scoring those points and temporarily disable the tagged robots motors.

Game Field

The game field is a set of tiles arranged into 4 continuous, non-intersecting winding lines. Each robot starts the game assigned to a specific line. At no point in the game is a robot allowed to drive completely off its line.





On each half of the board there are two lines that are interleaved. Each line has the same number of turns, but one line is longer. Each team has a long and short line on opposite sides of the field.

Game Rules

- Each game consists of a single 90 second match.
- A team scores 2 points for every time that it successfully travels the entire line from end to end while its beacon is active
- A team scores 1 point for every time that it "tags" an opponent robot while it has an activate beacon
- A team scores 5 points if both robots are at the end of their lines at the end of the game
- Any robot that drives completely off its line will be immediately removed from the game board
- A robot is considered off the line if no part of the line is underneath the robot.
- Robot Collisions
    - Robot to robot contact is considered a collision when robots contact sufficiently to cause at least one robot to drive off its line
    - 2 point penalty is given to the offending robot and both robots are reset and placed back in their starting position
    - The offending robot is the robot that was outside of its own tile when the collision occurred, if both robots are equally at fault then no penalty is given
- Beacon Overview
    - Each beacon is controlled by a simple 2 wire interface
    - Flag Request - this signal is sent from robot to beacon to activate the flag
    - Flag Carry / Drop - this signal is sent from the beacon to robot to indicate when it haslost its flag
- Beacon Penalty
    - Robots must always operate their beacons according to the game rules. Any beacon violation will result in a 2 point penalty, and the offending robot is reset and placed back in the starting position.
- Beacon Rules
    - A robot can only activate its beacon when it is located at the end of a line.
    - At the beginning of the game (or whenever its position is reset due to another rule violation) the robot must travel to the opposite end of the line before it can activate the beacon.
    - A robot is allowed to drop its beacon at any time, but it will only be given points if it keeps the beacon active for the entire length of the line.
    - A robot must remain disabled the entire time the flag drop signal is held low by the beacon

Hex Blitz

Overview

This is a competitive and cooperative challenge involving 2 teams of 2 robots per team. The objective of the game is to find and move balls on a hex grid playing field in order to score points.

Each team robots are allowed to move anywhere over its half of the playing field and can score points when a ball is successfully moved to the opposite side of the field over special scoring markers. At the end of the game each team receives additional points based on how many balls are on the opposite side of the field.

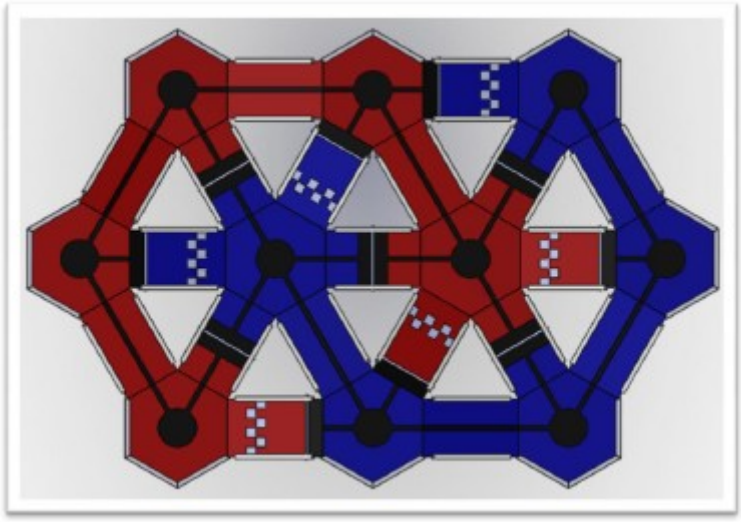
Coordinate between multiple robots to complete a task. Create and implement a strategy that maximizes the team's final score while following game rules

Game Field

The game field approximately 6 ft x 4 ft and is composed of 10 hexagon platforms connected with bridges and ramps. Four of the platforms are raised higher than the others. Each platform contains a black center marker and black lines leading to other platforms of the same team color or scoring areas for that team.

Platforms of different heights are connected with ramps, and platforms of the same height are connected with bridges. When a ramp connects opposite team zones, it is a scoring ramp and is marked with a checkered scoring area. A physical barrier at the intersection of opposing zones prevents a robot from accessing the opposite team's field areas but balls will pass under.

Walls surrounding every platform and connecting component prevent the robots and balls from leaving the playing area.



Game Rules

- Each game consists of a single 90 second match.
  - Robots that do not stop moving at the end of the 90 seconds may be penalized if their movement causes or prevents any points from scoring as balls come to rest.
- A team scores 1 points for every time that it passes a ball down a scoring ramp into the opponents side of the field.
  - Scoring ramps are the ramps that connect a high platform of one team to a low platform of the opposite team.
  - The point is scored when the ball crosses over the checkered scoring zone.
  - A defender is allowed to stop the ball from crossing over the score zone.
  - At the scoring ramp the zone barrier is at the high end of the ramp, allowing only the defending team access to the entire ramp, however, the slope of the ramp ensures a ball will always score unless a robot actively defends it.
  - No points are scored when a ball is pushed up the scoring ramp, by a defender. However, the points will be scored every time the ball passes completely through the score zone moving down the ramp

regardless of who last touched the ball.

- A team scores 2 points for every ball that is on the opposite side of the field at the end of the game.
  - Final ball positions are scored after all motion stops.
  - A ball touching one the black areas at zone barriers is not scored for either team.
  - Balls touching the black guide lines are scored the same as if they were touching the closest colored field area.
  - A team scores a 3 point bonus for every ball that is touching the black center marker on a platform at the end of the game.
- Any ball that leaves the playing field during the game will be returned to the field at the point closest to its exit location.
- Robot Collisions
  - Robots are expected to make routine, vigorous contact with all elements of the playing field, including the balls, zone barriers and walls.
  - If a robot's design or programming causes it to intentionally damage any game element or other robots, it will be disqualified.
  - Some robot to robot contact may occur when two opposing robots are at a zone barrier simultaneously. Generally, the extent of contact will be limited by the barrier itself, otherwise this is considered normal and fair play.
  - There is significant potential for robot collisions to occur within the same team. This must be considered by each team's programming strategy.
- At no point should a robot be touched by players after the game has begun.
  - A violation of this rule will result in the offending robot being immediately removed from game play.
  - A player may voluntarily remove their robot at any time, however, that robot will not be allowed to be returned to play during the game.
- Game Start
  - Robots must be positioned on the raised platforms at the start of the game (see Figure 2)
  - A robot can be placed anywhere on the platform, as long as some part of the robot is over the black center marker.
  - The 10 game balls will be arranged on each side of the zone barrier of every bridge as shown in Figure 2.
  - Robots should be designed to start the match when a sufficiently loud starting signal is detected (hand clap, whistle, buzzer, etc)
  - Robots should be designed to run autonomously for exactly 90 seconds.
  - If multiple false starts occur, the offending robot will have to be started manually at the buzzer. Normal end-of-game violations will be in effect even if there is a timer mismatch due to late manual starting.

Robot Modifications

- Robots are expected to be customized with ball manipulators, guards, etc.
  - Robots must have some physical mechanism that ensures the robot's body (metal frame, circuit board, etc) cannot move past a zone barrier.
  - A robot add-on is legal if it doesn't extend past the black zone marker in the middle of the bridges when the robot is in full contact with the barrier itself (this is about 1 1/2 inches).
  - Sufficient guarding should be added to protect a robot against contact with game elements and other robots.
- Robots can be extended with any kind of custom sensor
- Robots can be extended with additional motors or servos
- Robots can use any kind wireless communications for coordination between robots.
  - Absolutely no communication with non-robots should occur during any game
  - Unintentional interference of opposing team communications should be avoided
    - Teams using the same wireless technology should plan to coexist with other users of that technology (different channels, network ID, etc)
- Intentional interference of opponent team communication is not acceptable and violators will be disqualified.
  - The use of IR for proximity ranging on the robot is never considered interference even if the other team uses IR for communication.
- Any kind of monitoring of opponent team communications is legal and fair-play as long as such monitoring is completely passive.

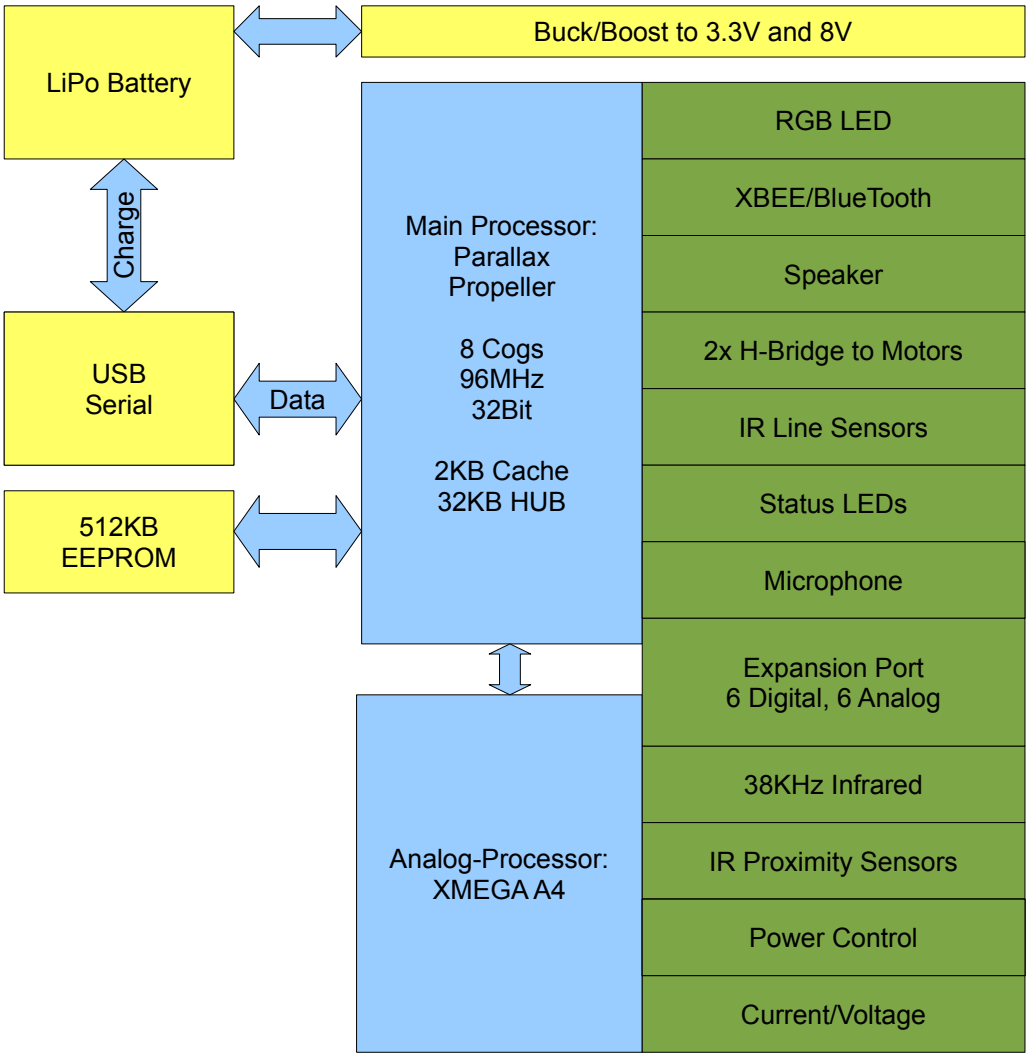
    - Here passive means that monitoring activity cannot have any measurable impact on the opposing team's communications.
    - If monitoring is determined to not meet the passive criteria, then it must be disabled or it is considered intentional interference.

Other Notes

- Most interior wall surfaces will be coated to intentionally minimize IR reflectance. This coating might just be black paint, but could be other textured materials that absorb and scatter IR light. This is to allow game balls to be the primary objects detected by IR proximity sensors.
    - The black guidance lines should be used as the primary means of navigation.
    - The scoring ramps are the exception to this rule, since there are no lines for guidance, the walls will be left white so they can be used for wall following.
- The balls will be standard 40mm matte white ping pong balls. Balls significantly damaged during matches will be replaced before the start of the next game, but playing with some minor damage should be expected.
- The zone coloring will be significantly lighter than it appears in the rendered images, however, teams should expect to need different line sensor calibration depending on which color they are playing as.
    - Teams will be allowed to run a quick calibration routine prior to placing robots on the field at the beginning of every match. These pre-match calibrations should take no more than 10-15 seconds.
    - Teams will not be allowed to recalibrate due solely to a false start (even if the restart is not their fault)
- A team is allowed to request the "Strict Robot Placement" rule if they decide its strategically important to them
    - Teams requesting this extra rule should inform the referee prior to starting the match
    - The referee will flip a coin to determine the first team placement: Head = Red, Tail = Blue
    - Then each team will alternate placing and positioning a robot on the field
    - After a robot is placed on the field, it cannot be re-positioned
    - Once this rule is put into effect, it remains in effect for all matches between those 2 teams
    - Without this rule, either team can reposition their robots as much as they want until both teams tell the referee they are ready to begin.

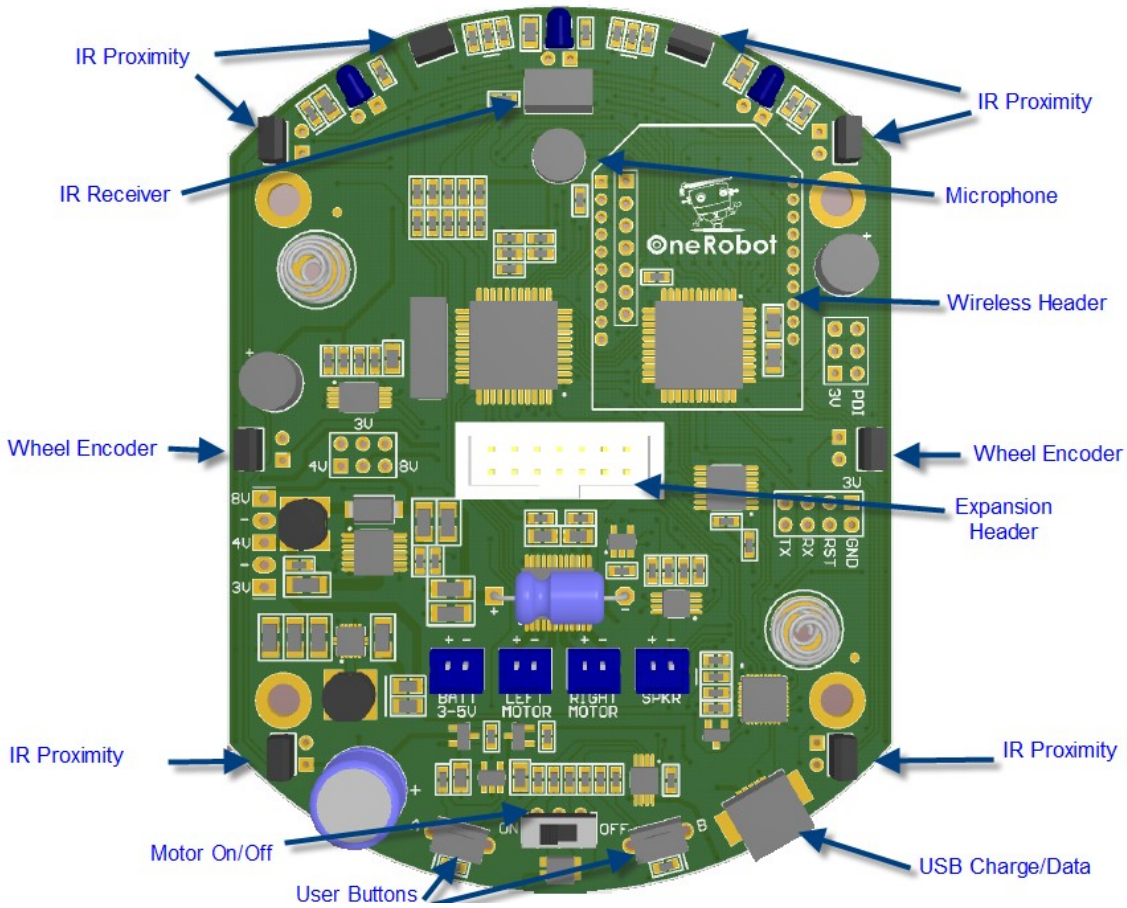
Chapter 6: Technical Details

Block Diagram



Programming Tool	**Interface**	**Components**	**Blocks**
12Blocks	USB	Propeller	start, repeat, set, if, functions
	Bluetooth	Motors	drive, turn, set speed, get voltage/current
Integrates with:	Xbee	Xbee	broadcast, terminal, mail
Skype	Wifi	Bluetooth	send, receive
ROS		Buttons	pressed, released, up, down
DDE		RGB LED	set color, luminosity, hue
Plugins		Line sensor	read
XMLRPC		USB	charge, read voltage
Fiducials		Proximity	get front/sides
		Sound	read mic, play wav, pluck, tone, speak
		Encoders	move with encoders

Features



Propeller running at 96MHz with 512K EEPROM

Programmable with 12Blocks or any other Propeller development tool.

ATXMEGA32A4

Running Tbot firmware developed with AVRStudio. 12Blocks includes a firmware loader that can upload custom firmware to the XMEGA using the Propeller.

5 line detectors sensors

Measures the reflectivity below the Tbot's front- great for high speed line following.

6 IR proximity sensors

Measures the distance to objects in the front and both sides by measuring how much IR light is reflected.

IR Transmitter/Receiver

Transmit and receive data using 38KHz IR pulses.

Powerful geared motors with encoders

Precise control and high speed.

Amplified speaker

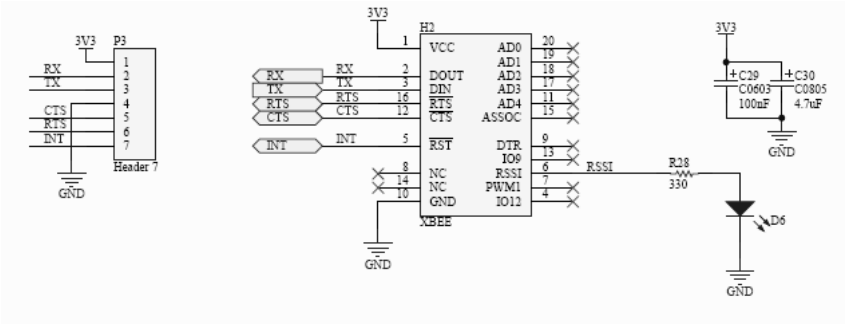
Top-mounted speaker controlled by Parallax Propeller. Can be programmed to play back sound files, synthesize speech, etc.

Microphone

React to sounds.

Wireless options

Tbot's PCB includes a header for XBee and BlueTooth modules- allowing full-duplex wireless programming and communication at high speed.



For more information on BlueTooth refer to: http://onerobot.org/guides/bluetooth.pdf

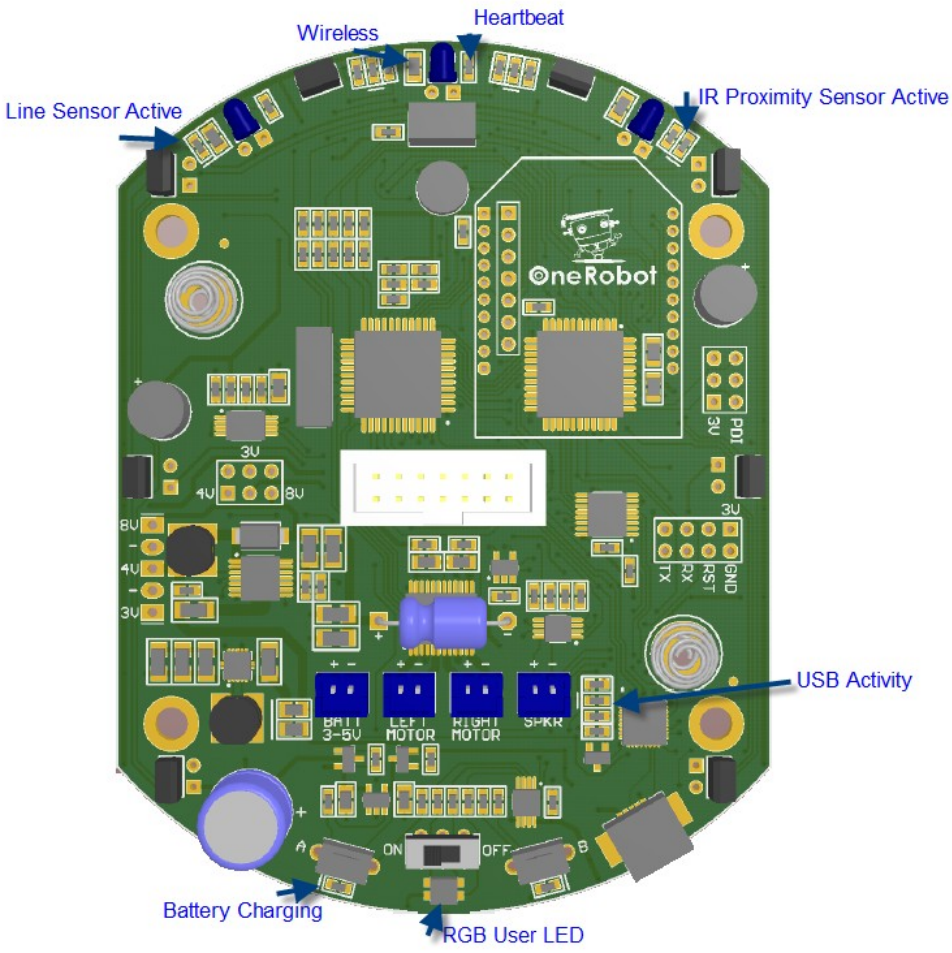
Full-color LED

Very bright full-color RGB LED

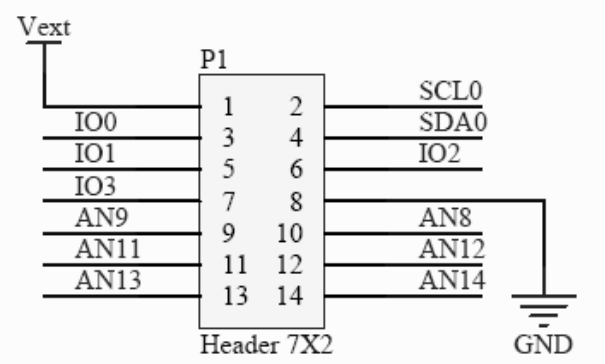
2 user switches

2 pushbuttons on rear for user input.

Status LEDs



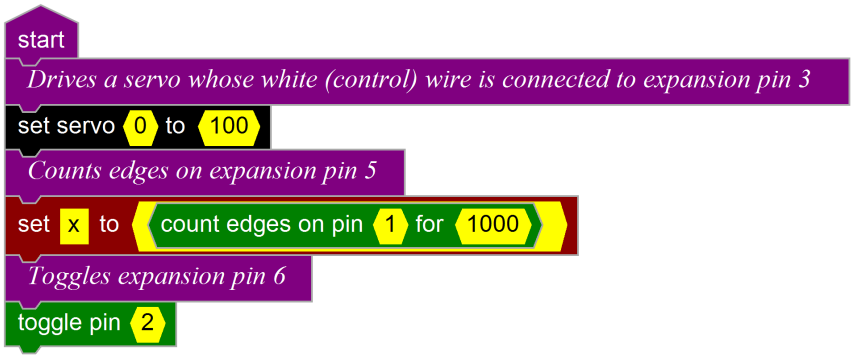
14 pin Expansion port with 6 hi-speed digital and 6 analog I/O



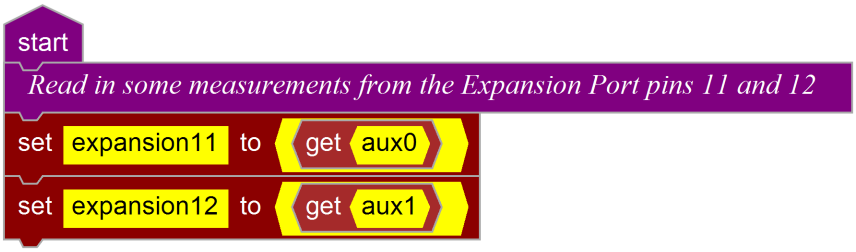
Pin	Detail
1	**Vext:** Power supply, same voltage, current capacity as driving wheels. Typically 8V, up to 1A
2	**SCL0:** Serial Clock of I2C protocol. Propeller P28
3	**IO0:** Digital IO, can be sampled/toggled at up to 96MHz. Source 3.3V at 50mA. Propeller P0
4	**SDA1**: Serial Data of I2C protocol. Propeller P29
5	**IO1:** Digital IO, can be sampled/toggled at up to 96MHz. Source 3.3V at 50mA. Propeller P1
6	**IO2:** Digital IO, can be sampled/toggled at up to 96MHz. Source 3.3V at 50mA. Propeller P2
7	**IO3:** Digital IO, can be sampled/toggled at up to 96MHz. Source 3.3V at 50mA. Propeller P3
8	**Ground**
9	**AN9:** Analog IO, XMEGA PB2
10	**AN8:** Analog IO, XMEGA PB3
11	**AN11:** Multiplexed Analog IO
12	**AN12:** Multiplexed Analog IO
13	**AN13:** Multiplexed Analog IO
14	**AN14:** Multiplexed Analog IO

A parallel hard drive connector can be adapted to connect to the central white 7x2 header.

To manipulate/sample the digital pins, use one of the blocks from the “pins” section of the library.



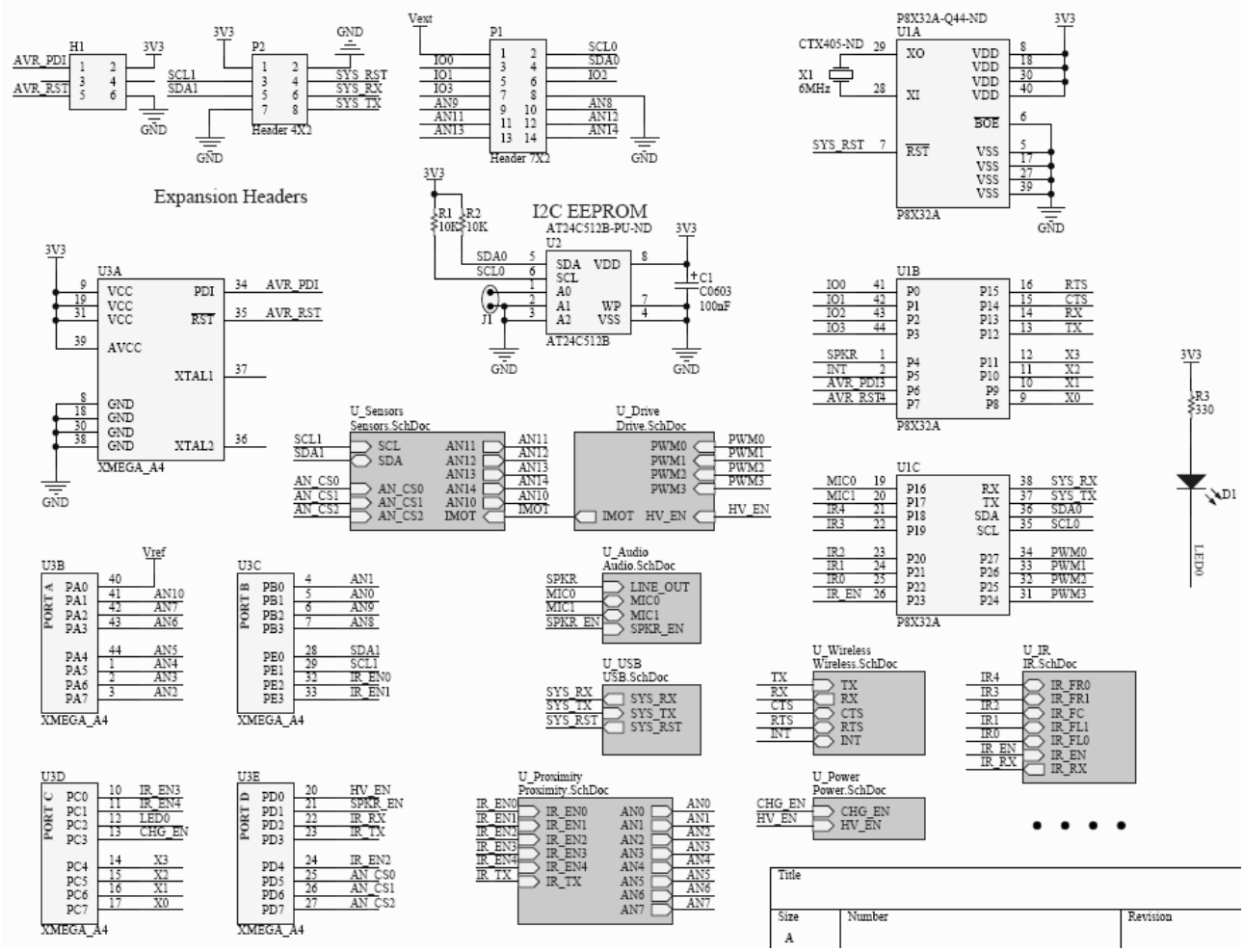
To sample Analog values, use the “get” block from the “TBot” library.

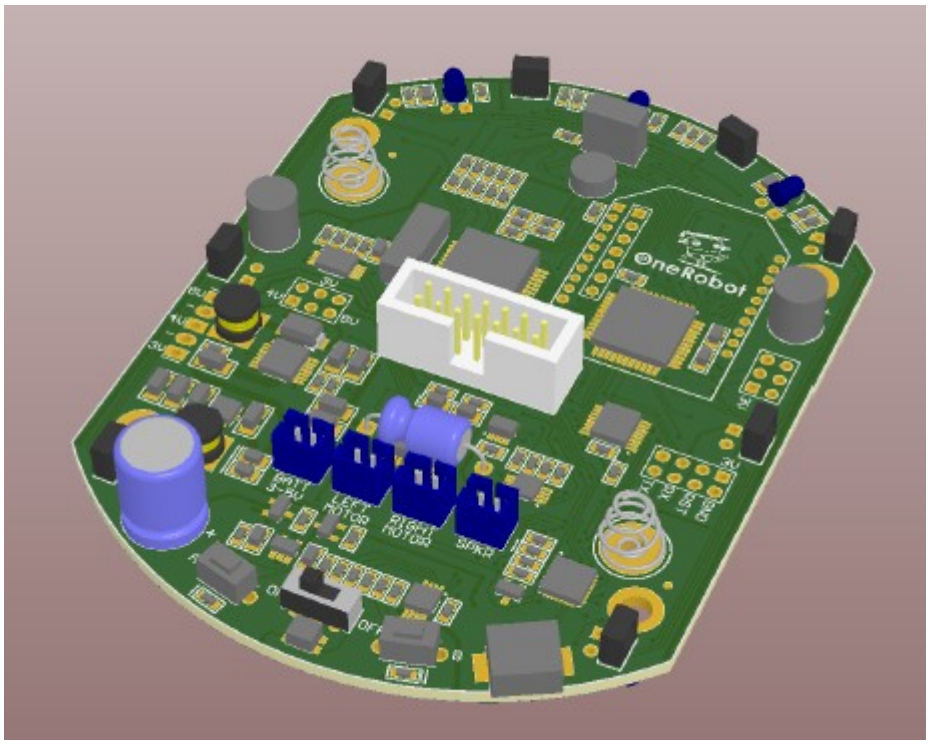


Li+ Battery charges via USB

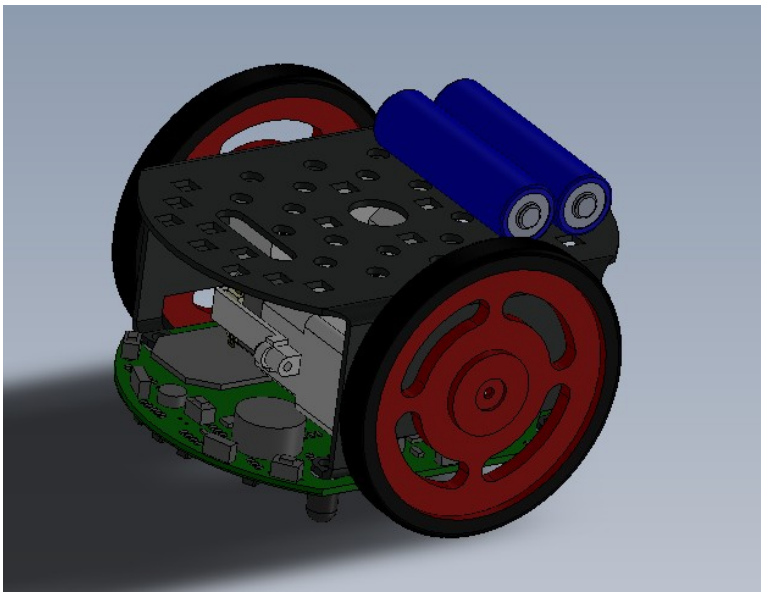
TBot includes the FTDI USB to serial chip, so you just need a USB cable- not a PropPlug.
To charge Tbot's battery just plug it into a USB cable connected to a PC.

Schematics





Mechanics



Instruction Set

Tab	Section	Name	Detail
control	basic	start	start running a set of blocks
		repeat	continually run the inner blocks
		repeat (TIMES) times	run the inner blocks a number of times TIMES: number of times to loop
		repeat (VAR) from (START) to (END) step (STEP)	run the inner blocks with specified variable, start, end and steps VAR: loop variable which counts from start to stop START: the loop variable will start with this value END: the loop variable will end with this value STEP: the loop variable will change by this amount each time
		repeat while (CONDITION)	run the inner blocks while the condition is true CONDITION: when this condition is true, the inner stacks will continue to run
		repeat until (CONDITION)	run the inner blocks until the condition is true CONDITION: when this condition is true, the inner stacks will continue to run
		wait (DURATION)	pause for a duration DURATION: time in milliseconds
		if (CONDITION) else	runs the first inner blocks if the condition is true, otherwise runs the second inner blocks CONDITION: when this condition is true, the inner blocks will run
		case (VALUE)	runs the inner block whose value matches the condition VALUE: value to match *CONDITION*: when this condition is true, the inner blocks will run *Right click on block to access Properties for custom parameters
	advanced	show/edit program info	display info about variables, arrays and imports. Ctrl-click on imported files to open them *POSITION*: position of block *SIZE*: size of block *LOOK*: look of block *Right click on block to access Properties for custom parameters
		break out of loop	break out of a loop
		continue to start of loop	continue to start of loop
		stop	stop this program
		(COMMENT)	to document your code COMMENT: comment which explains your code
		(COMMENT)	write text code COMMENT: code
	state	when in state (STATE NAME)	start running a set of blocks when in a state

			STATE NAME: state which starts this stack of blocks
		run state machine (NAME OF STATE MACHINE)	run a state machine using a variable NAME OF STATE MACHINE: variable to use for storing machine's state
		set state to (STATE NAME)	set the current state of this machine STATE NAME: the state machine's new state
	event	when (KEY)	run a set of blocks when a condition is true KEY: condition which starts this stack of blocks
		task (TASK NAME)	start running a set of blocks when a message is received TASK NAME: name of task
		start task (TASK NAME)	start a named task and immeadiately continue TASK NAME: task to run
		start task (TASK NAME) and wait	start a named task and wait for it to finish before resuming TASK NAME: task to run
tbot		broadcast (VARIABLES)	stream variables to all bots VARIABLES: variables to broadcast *START*: *END*: *Right click on block to access Properties for custom parameters
		receive (VARIABLES)	receive variables from other bots VARIABLES: variables to receive *START*: *END*: *Right click on block to access Properties for custom parameters
		start terminal session with bot (BOT) using (QUEUE)	start terminal session with bot $ using $ BOT: bot to communicate with QUEUE: terminal queue
		stop terminal session (TERMINAL)	stop terminal session $ TERMINAL: terminal session
		send (BYTE) with (TERMINAL)	send a byte BYTE: byte to send TERMINAL: terminal session
		receive from (TERMINAL)	receive a byte TERMINAL: terminal session
		mail (OUTARRAY) to bot (BOT) respond into (INARRAY)	send a message OUTARRAY: message to send BOT: bot to send to INARRAY: message to receive
		reply to (BOT) with (OUTARRAY)	reply to a message BOT: bot to send to OUTARRAY: message to send
		receive mail from (BOT) into (MAIL)	reply to a message BOT: bot to send to MAIL: mail bag
		receive bluetooth	check for received character
		send bluetooth string((VALUE))	send bluetooth string VALUE:

		send bluetooth number((VALUE))	send bluetooth number VALUE:
		get bluetooth character	get next character
		get (SENSOR)	get $ SENSOR:
		(WHICH BUTTON) button (BUTTON STATE) within (TIMEOUT) msec	wait until a button event or timeout occured WHICH BUTTON: button to monitor BUTTON STATE: button state TIMEOUT:
		(WHICH BUTTON) button (BUTTON STATE)	returns button status WHICH BUTTON: button to monitor BUTTON STATE: button state
		set LED (HUE) (SATURATION) (LUMINOSITY)	set the color led HUE: SATURATION: LUMINOSITY:
		read line sensor((BUTTON))	read line sensor($) BUTTON: line sensor to read
terminal		print text (TEXT)	send text to the terminal TEXT: text to send
		print value (VALUE)	send a value VALUE: value to send
		clear screen	clear the terminal screen
		backspace	go back one space
		next line	go to the next line
		goto (X),(Y)	go to the specified point X: x position of cursor Y: y position of cursor
		receive text into (ARRAY)	receive text from the terminal and store in an array ARRAY: array into which to store received text
		receive number	receive a number from the terminal
		receive byte	receive a byte from the terminal
		data available	test if the terminal has sent something
sound		play synthesized song (SOUND)	play a synthesizer file SOUND: name of hmus file to play
		stop playing song/effect	stop playing song/effect
		play sound effect (SOUND)	play sound effect $ SOUND: name of hsfx file to play
		play score (SCORE)	play score $ SCORE: score to play
		pluck score (SCORE)	pluck score $ SCORE: score to play
		set pluck volume to (VOLUME) tempo to (TEMPO) sustain to (SUSTAIN)	change the pluck volume, tempo and sustain VOLUME: volume to pluck at TEMPO: tempo of plucking SUSTAIN: sustain of pluck
		play wav file (SOUND) at volume (VOLUME)	play a wav file SOUND: name of wav file to play VOLUME: volume to play file at

		record sound for (DURATION) ms	record a sound to memory DURATION: milliseconds to record sound
		play recorded sound at volume (VOLUME)	play back recorded sound from memory VOLUME: volume to play sound at
		read microphone	read microphone
		play tone (FREQUENCY) for (DURATION) ms	play a musical tone FREQUENCY: frequency of tone DURATION: milliseconds to play tone
		set tone volume to (VOLUME)	change the tone volume VOLUME: volume to play tone at
		speak (SPEECH)	speak text SPEECH: text to say
		speak file (SPEECH)	speak a word SPEECH: text to say
		spell (SPEECH)	spell text SPEECH: text to say
		set speech volume to (VOLUME)	change the speech volume VOLUME: volume to use for speech
		speech parameters (GLOTTAL PITCH), (VIBRATO PITCH),(VIBRATO RATE), (PACE)	change the way words are spoken GLOTTAL PITCH: voice pitch, 100=110hz VIBRATO PITCH: voice vibrato pitch, 48=+/- half octave swing VIBRATO RATE: voice vibrato rate, 52=4Hz PACE: rate at which word is spoken
		set speaker (SPEAKER) to pitch (BASE)	Assign a base pitch to speaker number SPEAKER: BASE:
motion		move with encoders (LEFT) (RIGHT)	go with encoders LEFT: RIGHT:
	motors	move (TIME)	move forwards TIME: msec to drive robot, negative to reverse
		turn (TURN)	turn TURN: msec to turn the robot
		smooth turn (TURN)	turn TURN: msec to turn the robot
		set motors (LEFT) (RIGHT)	turn LEFT: rate to move the robot RIGHT: rate to move the robot
		lock motor (LEFT) (RIGHT)	turn LEFT: RIGHT:
		set cruise speed (CRUISE)	set cruise speed CRUISE:
		correct motors by (STOP LEFT),(STOP RIGHT),(GAIN LEFT),(GAIN RIGHT)	correct for motors with differnt speeds and 0 points STOP LEFT: STOP RIGHT: GAIN LEFT: GAIN RIGHT:
	servos	set servo (SERVO) to (POSITION)	set a servo's position SERVO: pin number of servo POSITION: target position for servo

		ramp servo (SERVO) to (POSITION)% over (TIME)	move a servo to a new position over time SERVO: pin number of servo POSITION: target position for servo, from -30 to 130 TIME: milliseconds to ramp the servo to the new position
		idle servo (SERVO)	idle a servo- don't power it SERVO: pin number of servo
sense	time	reset timer	reset the internal timer
		elapsed time	read the elapsed time in ms since last reset
vars		random((MAX))	random MAX:
		pid response to error (ERROR) with p=(P) i=(I) d=(D)	pid control block ERROR: P: I: D:
		restrict (VAR) between (MIN) and (MAX)	restrict $ between $ and $ VAR: MIN: MAX:
	variables	set (VARIABLE) to (VALUE)	set a variable to a value VARIABLE: variable to set VALUE: new value for variable
		change (VARIABLE) by (DELTA)	change a variable's value VARIABLE: variable to change DELTA: amount added to variable
		set bit (BIT) of (VARIABLE) to (NEW BIT VALUE)	set a bit BIT: bit to set VARIABLE: variable to modify NEW BIT VALUE: bit value
		get bit (BIT) of (VARIABLE)	get a bit BIT: bit to get VARIABLE: variable to inspect
	arrays	set (ARRAY)((INDEX)) to (VALUE)	set an array's item to a value ARRAY: array to set INDEX: index of array item to set VALUE: new value for array item
		change (ARRAY)((INDEX)) by (DELTA)	change an array's item ARRAY: array to change INDEX: index of array item to change DELTA: amount added to array item
		get (ARRAY)((INDEX))	get an array's item ARRAY: array to retrieve INDEX: index of array item to retrieve
	strings	set (ARRAY) to (STRING)	set $ to $ ARRAY: array to modify STRING: text to assign
		set (TEXT) to value (NEW VALUE)	set $ to value $ TEXT: array NEW VALUE: value to add
		string (FIRST) equals (SECOND)	string $ equals $ FIRST: first text/array to compare SECOND: second text/array to compare

lowercase (TEXT)	lowercase $ TEXT: array to modify
uppercase (TEXT)	uppercase $ TEXT: array to modify
capitalize (TEXT)	capitalize $ TEXT: array to modify
reverse (TEXT)	reverse $ TEXT: array to modify
make (COUNT) copies of (TEXT)	make $ copies of $ COUNT: number of copies to make TEXT: array to modify
trim (TEXT)	trim $ TEXT: array to modify
pad (TEXT) to length (LENGTH) with (PAD)	pad $ to length $ with $ TEXT: array to modify LENGTH: length to pad to PAD: text/array to pad with
replace (REPLACEE) with (REPLACER) in (TEXT)	replace $ with $ in $ REPLACEE: old text REPLACER: new text TEXT: array to modify
join (NEW TEXT) to (TEXT)	join $ to $ NEW TEXT: text/array to add TEXT: array join to
put (ITEM) split of (TEXT TO SPLIT) into (RESULT)	put $ split of $ into $ ITEM: item of split TEXT TO SPLIT: text/array to split RESULT: result string
join (NEW BYTE) to (TEXT)	join $ to $ NEW BYTE: byte to add TEXT: array join to
get character (TEXT)((INDEX))	get character $($) TEXT: text/array from which to get a character INDEX: index
copy substring from (TEXT) starting at (START) for (COUNT) to (OUTPUT)	copy substring from $ starting at $ for $ to $ TEXT: text/array from which to make substring START: starting index COUNT: characters to copy OUTPUT: output array
copy string beginning with (BEGIN) in (TEXT) starting at (START) to (OUTPUT)	copy string beginning with $ in $ starting at $ to $ BEGIN: text/array to find TEXT: text/array to search in START: starting index OUTPUT: output array
find index of string (STRING) in (TEXT) starting at (START)	find index of string $ in $ starting at $ STRING: string to find TEXT: text/array to search START: starting index
find first index of (CHARACTER) in (TEXT) starting at (START)	find first index of $ in $ starting at $ CHARACTER: character to find TEXT: text/array to search START: starting index

		find last index of character (CHARACTER) in (TEXT) starting at (START)	find last index of character $ in $ starting at $ CHARACTER: character to find TEXT: text/array to search START: starting index
		length of (TEXT)	length of $ TEXT: text/array to count
		convert (TEXT) to a number in base (BASE)	convert $ to a number in base $ TEXT: text/array to search BASE: use 10 to convert to decimal
interface		background	draw a background image *POSITION*: position of background *SIZE*: size of background *LOOK*: look of block *Right click on block to access Properties for custom parameters
		skype	communicate with skype *POSITION*: position of block *VARLIST*: list of variables to share with skype *SIZE*: size of block *LOOK*: color/images of block *FORMAT*: format string *SETPOINTS*: coordinates of text within block *Right click on block to access Properties for custom parameters
		xmlrpc	communicate with xmlrpc *POSITION*: position of block *VARLIST*: list of variables to share with xmlrpc *SIZE*: size of block *LOOK*: color/images of block *URI*: uri of xml-rpc server *SETPOINTS*: coordinates of text within block *Right click on block to access Properties for custom parameters
		ros	communicate with ros *POSITION*: position of block *VARLIST*: list of variables to share with ros *SIZE*: size of block *LOOK*: color/images of block *URI*: uri of xml-rpc server *SETPOINTS*: coordinates of text within block *Right click on block to access Properties for custom parameters
		fiducials	control with computer vision fiducials *POSITION*: position of block *VARLIST*: list of variables to share with fiducial engine *SIZE*: size of block *LOOK*: color/images of block *SETPOINTS*: coordinates of text within block *Right click on block to access Properties for custom parameters
		textbox	display and change a variable's value with a

			textbox *POSITION*: position of textbox *VAR*: variable to display and change *VALUE*: new value for variable *SIZE*: size of block *LOOK*: color/images of block *SETPOINTS*: coordinates of text within block *CAPTION*: caption for block *Right click on block to access Properties for custom parameters
		meter	display a variable's value in a meter *POSITION*: position of meter *VAR*: variable to meter *MIN*: minimum value of variable *MAX*: maximum value of variable *SIZE*: size of block *LOOK*: color/images of block *SETPOINTS*: rectangle coordinates of meter within block *CAPTION*: caption for block *Right click on block to access Properties for custom parameters
		switch	display and change a variable's value as a switch *POSITION*: position of switch *VAR*: variable to switch *SIZE*: size of block *LOOK*: color/images of block *CAPTION*: caption for block *Right click on block to access Properties for custom parameters
		joystick	use a joystick to control two variables *POSITION*: position of joystick *VARX*: x-variable for joystick *VARY*: y-variable for joystick *MIN*: minimum value or joystick variables *MAX*: maximum value for joystick variables *SIZE*: size of block *LOOK*: color/images of block *SETPOINTS*: coordinates and size of stick within block *CAPTION*: caption for block *Right click on block to access Properties for custom parameters
		save	save to file *POSITION*: position of block *VARLIST*: list of variables to save *SIZE*: size of block *LOOK*: color/images of block *SETPOINTS*: coordinates of text within block *FORMAT*: format string *FILE*: file to append data to *Right click on block to access Properties for custom parameters
		gamepad	use a joystick to control two variables *POSITION*: position of gamepad *PORT*: USB input device to monitor

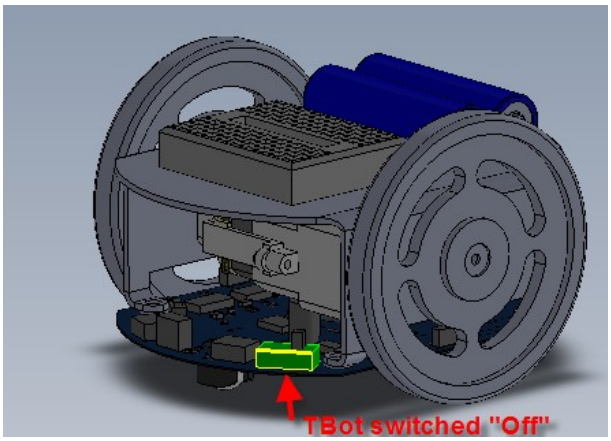
			STYLE: Type of attached gamepad: logitech, xbox *VARX1*: x-variable for left thumbstick *VARY1*: y-variable for left thumbstick *VARX2*: x-variable for right thumbstick *VARY2*: y-variable for right thumbstick *VARBTN*: variable for button states *VARPOV*: variable for pov direction *MIN*: minimum value for thumbstick axis *MAX*: maximum value for thumbstick axis *SIZE*: size of block *LOOK*: color/images of block *SETPOINTS*: coordinates and size of stick within block *Right click on block to access Properties for custom parameters
functions		(NAME) ((ARGUMENTS)) locals: (LOCALS)	define a user function NAME: name of your function ARGUMENTS: arguments passed by to this function LOCALS: local variables to this function
		return (RETURN VALUE)	return a value from a function RETURN VALUE: value to return
pins	in	count edges on pin (PIN) for (DURATION)	count the number of rising edges on a pin PIN: pin number to count edges on DURATION: milliseconds during which edges are counted
		measure frequency on pin (PIN) for (DURATION)	measure the frequency on a pin PIN: pin number to measure frequency on DURATION: milliseconds during which frequency is measured
		measure pulse on pin (PIN) at state (STATE)	measure the duration of a pulse a pin PIN: pin number to measure pulse on STATE: state of pin
		pin (PIN)	read the state of an IO pin PIN: pin on which state is measured
		duration of discharge on pin (PIN)	measure the time until a pin's state changes PIN: pin which is tested
		shift data in from pin (PIN) mode (MODE)	shift data into a pin from another device PIN: pin from which data is shifted in MODE: mode
		serial in from pin (PIN) mode:((BAUD),(MODE),(BITS))	read data using the serial protocol PIN: pin from which data is received BAUD: rate at which data is received MODE: mode: 0=inverted(normally low) 1=non-inverted(normally high) BITS: number of bits to receive
		read i2c on pin (PIN) and reply with (ACKBIT)	read byte using i2c protocol and acknowledge PIN: pin to transmit on ACKBIT: acknowledge bit
		read (BYTES) bytes from (ADDRESS) into (DATA)	read data from an i2c eeprom BYTES: bytes to write ADDRESS: eeprom address DATA: data array
	out	output frequency (FREQUENCY) on pin (PIN)	continually output a frequency on a pin FREQUENCY: frequency to output in Hz

			PIN: pin for output
		output frequency (FREQUENCY) on pin (PIN) for (DURATION)	output a frequency on a pin for a duration FREQUENCY: frequency to output in Hz PIN: pin for output DURATION: milliseconds for output
		set pin (PIN) high	set a pin high PIN: pin to set high
		set pin (PIN) low	set a pin low PIN: pin to set low
		toggle pin (PIN)	change a pin's state from high to low/low to high PIN: pin to change
		output pulse length (DURATION)uSec on pin (PIN)	output a pulse DURATION: microseconds to output pulse PIN: pin to output
		output pwm (DUTY) on pin (PIN)	output a pulse width modulated signal DUTY: duty cycle, form 0 to 256 PIN: pin to output
		output pwm (DUTY) on pin (PIN) for (DURATION)	output a pulse width modulated signal DUTY: duty cycle, form 0 to 256 PIN: pin to output DURATION: milliseconds for output
		shift out data (DATA) on pin (PIN) mode (MODE)	shift data to a device DATA: value to shift out PIN: pin to output to MODE: mode
		send serial data (DATA) on pin (PIN) mode: ((BAUD),(MODE),(BITS))	send data with the serial protocol DATA: value to transmit PIN: pin to transmit on BAUD: rate at which data is transmitted MODE: mode: 0=inverted(normally low) 1=non-inverted(normally high) BITS: bits to transmit
		initialize i2c device on (PIN)	initialize the i2c device PIN: i2c scl pin
		send start i2c token on (PIN)	send a start i2c token PIN: i2c scl pin
		write i2c data (DATA) to pin (PIN)	write data with the i2c protocol DATA: value to transmit PIN: pin to transmit on
		send stop i2c token on (PIN)	send a stop i2c token PIN: i2c scl pin
		write (BYTES) bytes of (DATA) to (ADDRESS)	write data to an i2c eeprom BYTES: bytes to write DATA: data array ADDRESS: eeprom address
		share array: (ARRAYS)	share arrays with 12Blocks to upload/download data ARRAYS: type a quoted, commas separated list of array names
		quickly sample the IO pins	quickly sample the IO pins

Chapter 7: Troubleshooting

If you're having trouble with your TBot, please ensure the following:

- 12Block is installed on computer- for help see "Installation Guide"
- USB cable is connected to computer and TBot
- TBot is charged and switch is in middle position "Run without motors"
- Write or load a program in 12Blocks with the TBot library
- Quickly run a program by pressing "Run" in 12Blocks- keep cable connected
- Motors only run if switch is in "Run with motors" position
- Permanently load a program from "Device" menu- can then disconnect cable and restart by switching "off" and "on"
- Turn TBot "off" after use, LED's should turn off.



Appendix A: Optional Parts

To complete the activities in this text, you will need a complete TBot robot.

For the latest information, downloads, and accessories, visit www.OneRobot.org

Aside from a PC with a serial or USB port and a few common household items, the TBot Robot Kit contain all the parts and documentation you'll need to complete the experiments in this text.