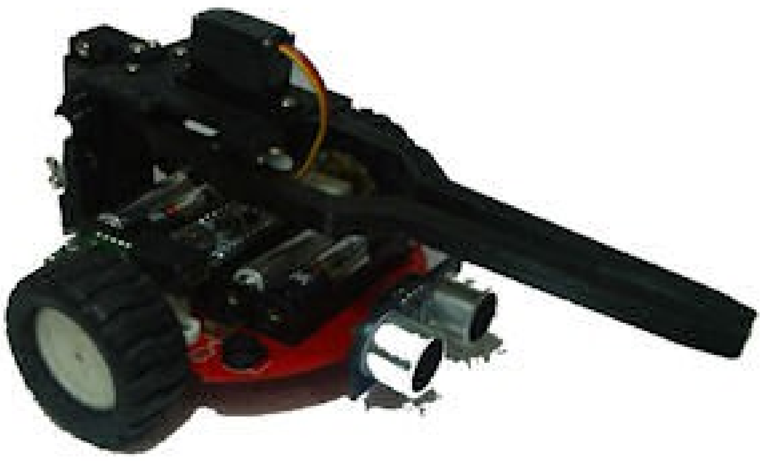
Learning with TBot

Version .4

1



Warranty

14-Day Money Back Guarantee

Copyrights And Trademarks

Disclaimer Of Liability

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 motor control, 2-axis gripper, optional LCD display, 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://onerobot.org/forums/ 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. Curriculum targeting different age groups are available online at http://onerobot.org/ols/ols.php

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 Author

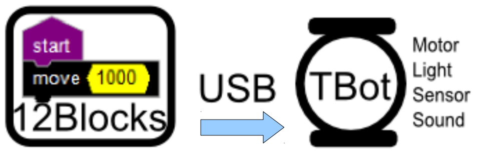
Hanno Sander earned a degree in Computer Science from Stanford University, where he built one of the first hybrid cars, collaborated on a microsatellite, and studied artificial intelligence. He later founded a startup to develop customized information services and then transitioned to product marketing 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: Dave Westbrook, Ingolf Sander, Professor Tim Bell, Steve Woodrough

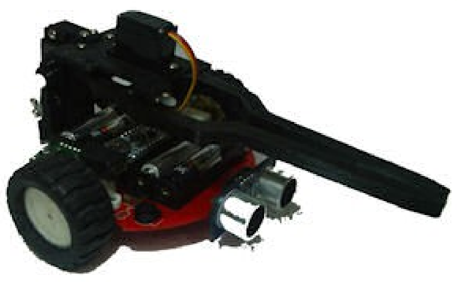
Chapter 1: Getting Started

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.

A powerful Arduino microcontroller control your TBot's motors, lights, sounds, grippers, 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 USB male connector to the front of your TBot. Finish by connecting the USB cable to your PC.

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's. To run the TBot's motor and gripper for movement, you'll need to slide the power switch on the bottom to "ON".

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

Step 2: Install Software

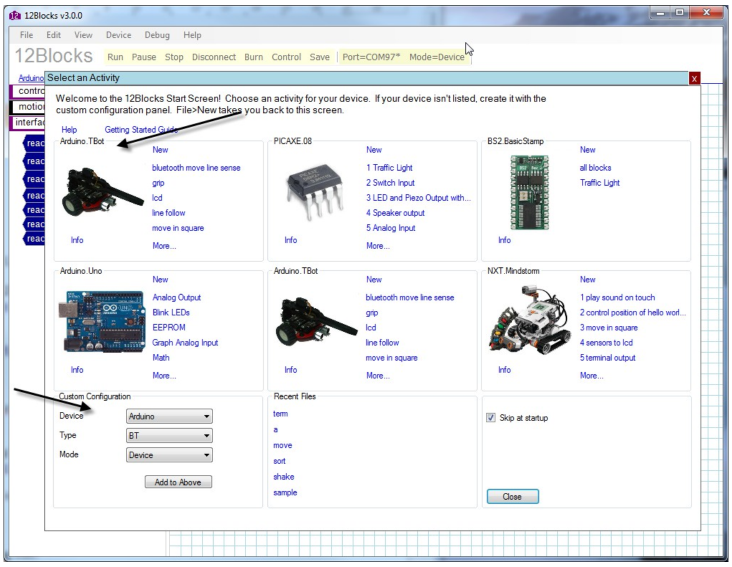
12Blocks is a visual and easy to use drop-and-drag programming language that is highly suitable for educational purposes. It is mainly targeted at programming robots and microcontrollers and supports various types of devices.

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 a start screen. If "Arduino.TBot" is not shown in the first 2 lines, add it using the "Custom Configuration" in to bottom left.

Click on a TBot activity or "New" to get started with robotics!



Welcome to the 12Blocks Start Screen! Choose an activity for your device. If your device isn't listed, create it with the custom configuration panel. File>New takes you back to this screen.

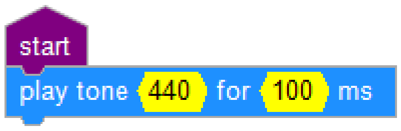
Custom Configuration

Device	Arduino
Type	BT
Mode	Device

[✓] Skip at startup

Step 3: Your First Program

For our first program we'll program the TBot to beep when it's button is pushed. Drag the blocks to the worksheet and press the "Run" command in the toolbar.



Concepts:

– Use a "start" block from the "controls" library to begin a stack of blocks
– Drag the "play tone" block" from the "sound" library and attach it to the "start" block.
– Click on the yellow part of a block to change a parameter.

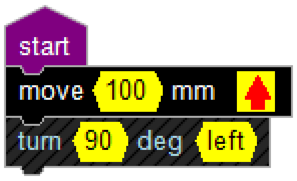
Hints and Tips

- 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
- Programs loaded to the TBot are stored there until the next program is loaded. You can disconnect the USB cable, turn-off the robot, and when you turn it on again it'll run your program from the start.
- 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.



```
start
move 100 mm
turn 90 deg left
```

Concepts:

- “File>New” to start a new program
- “File>Save” to save it
- “move” and “turn” are in the “motion” section

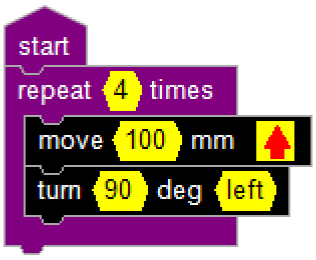
Navigation

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.

Square Dancing

In the last program the TBot made 2 moves. Let's improve that by using the “repeat x” block to repeat a set of blocks a number of times to move in a square.

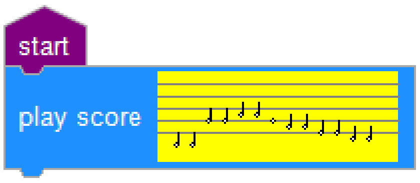


```
start
repeat 4 times
    move 100 mm
    turn 90 deg left
```

Music

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

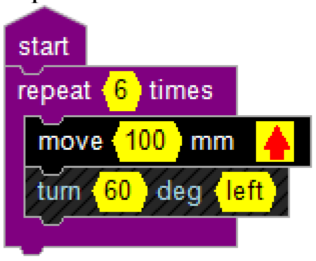
The “sound” section contains different blocks to make sounds.



```
start
play score
```

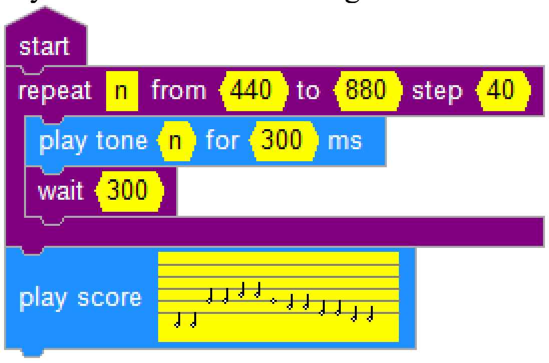
More Sample Programs:

1. Move in a square



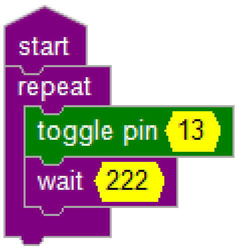
```
start
repeat 6 times
    move 100 mm
    turn 60 deg left
```

2. Play musical notes then song



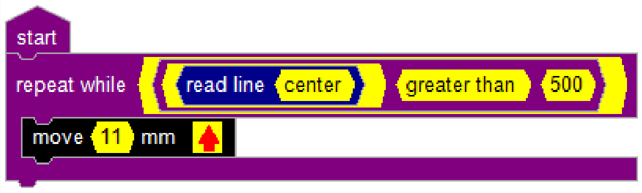
```
start
repeat n from 440 to 880 step 40
    play tone n for 300 ms
    wait 300
play score
```

3. Flash LED



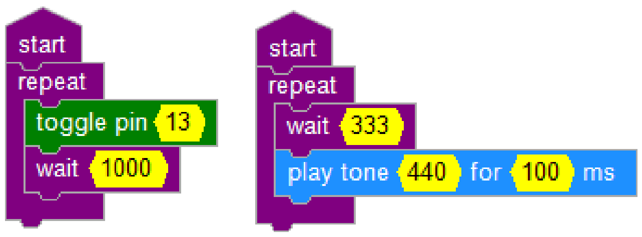
```
start
repeat
    toggle pin 13
    wait 222
```

4. Move on black surface



```
start
repeat while ((read line center) greater than 500)
    move 11 mm
```

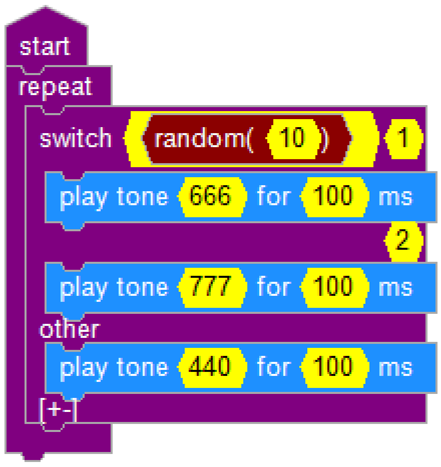
5. Blink led and play sound at the same time



```
start
repeat
    toggle pin 13
    wait 1000
```

```
start
repeat
    wait 333
    play tone 440 for 100 ms
```

6. Make random sounds

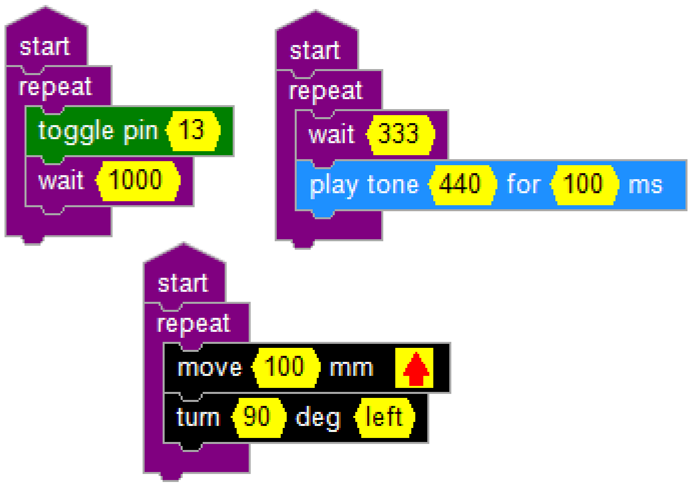


```
start
repeat
    switch (random( 10 )) 1
        play tone 666 for 100 ms
    2
        play tone 777 for 100 ms
    other
        play tone 440 for 100 ms
    [+-]
```

Chapter 3: Advanced Programs

Multiprocessing

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



```
start
repeat
    toggle pin 13
    wait 1000

start
repeat
    wait 333
    play tone 440 for 100 ms

start
repeat
    move 100 mm
    turn 90 deg left
```

Chapter 4: Sample Classwork

Lab 1 - Introduction to TBot

- Learn to use tool chain to write programs and load into robot
- Use user buttons to control program execution
- Read Sensors (Proximity, Microphone)
- Drive motors under basic sensor control
- Use LED/Speaker for representing robot status

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

Lab 3 - Maze Solver

- Learn to utilize multiple control techniques to complete a task
- 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 competitions 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

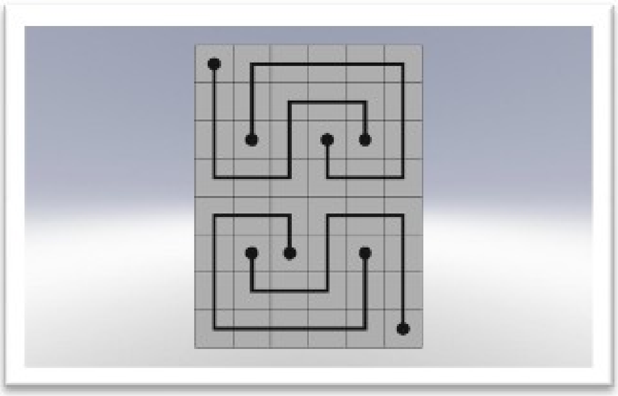
- Coordinate between multiple robots to complete a task
- Develop a protocol to handle communication between robots
- Create a strategy that maximizes the team's final score while following game rules

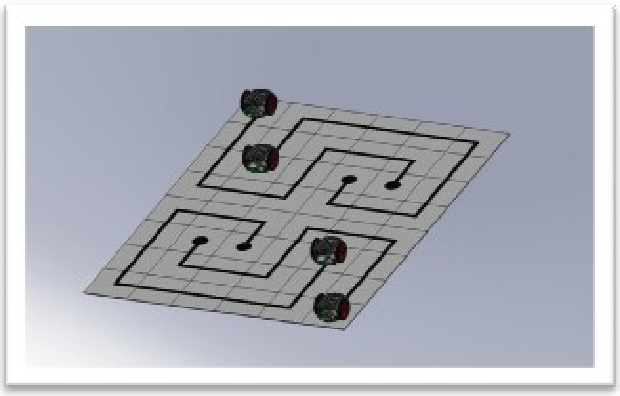
This is a competitive and cooperative challenge involving 2 teams of 2 robots per team. The objective of the game is follow a winding "maze-like" line from one end to the other in order to score points. Each robot will be equipped with a special beacon that it can activate when it reaches one end of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

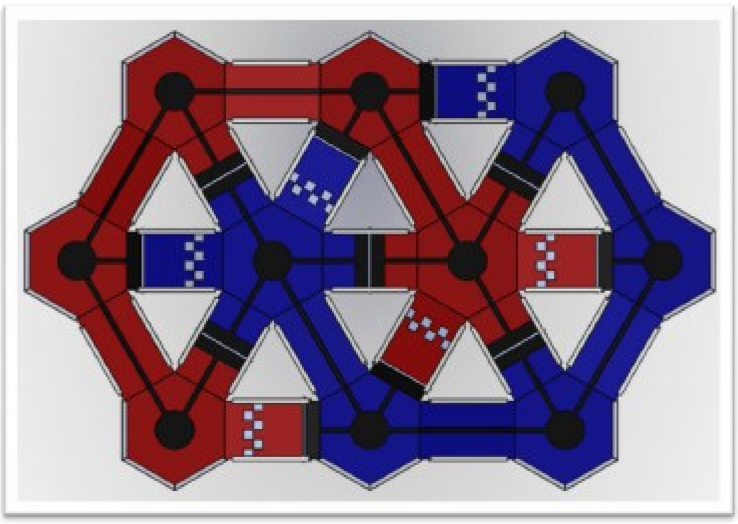
Coordinate between multiple robots to complete a task. Create and implement a strategy that maximizes the team's final score while following game rules

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

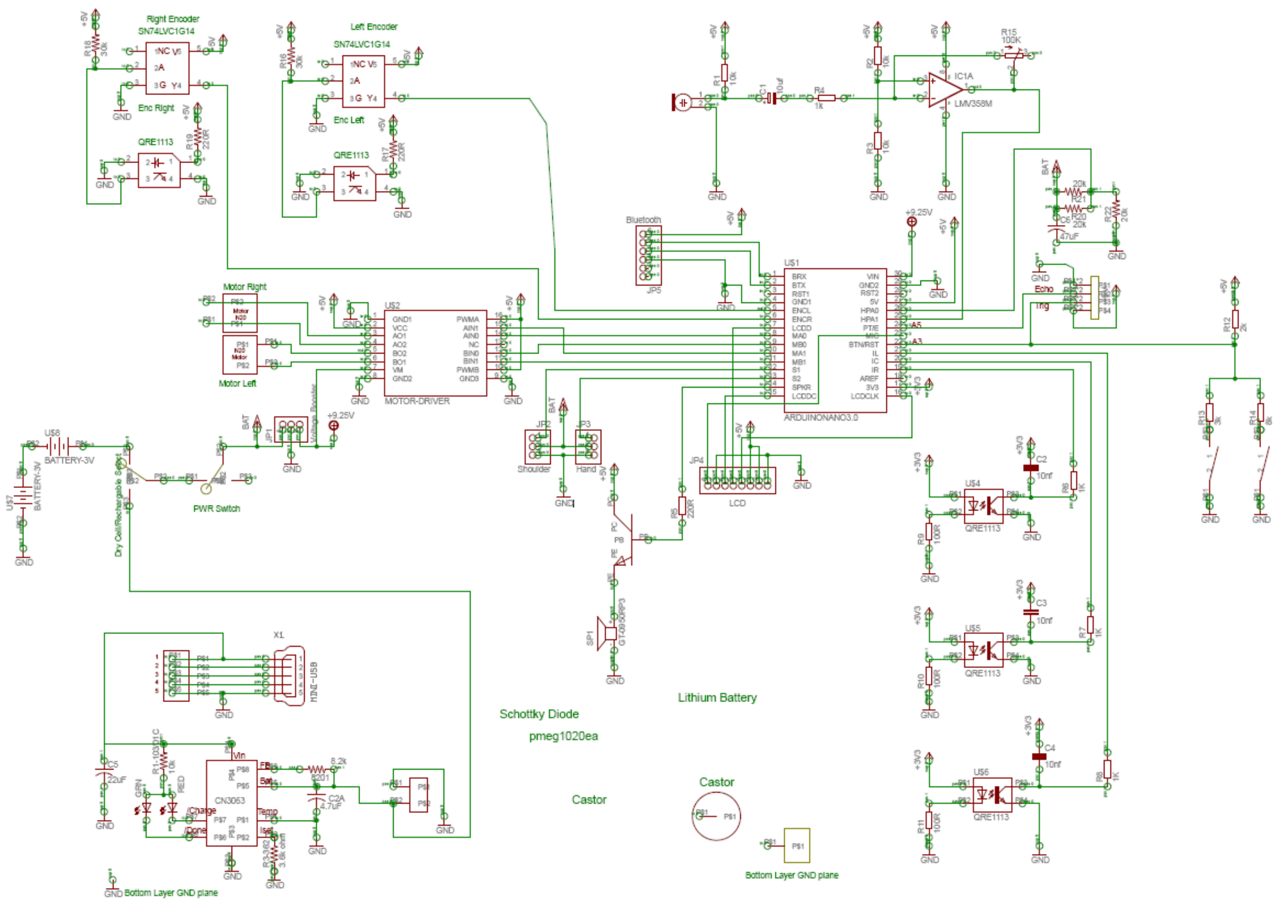
- Line detectors, microphone, ultrasound proximity, buttons, LEDs
- Two-axis grippers to lift and grip objects
- Fast geared motors with encoders to move 1 m/sec with accuracy of 1 mm.
- Amplified speaker and graphical LCD
- Fully assembled and ready to run out of the box.
- Included lithium battery charges when connected to USB
- Powerful Arduino processor
- Programmable with 12Blocks and C code

Pin Connections to Arduino Nano

Controller Overview

Line Sensor R	A0		0,1	Bluetooth
Line Sensor C	A1		2,3	Encoder L,R
Line Sensor L	A2		4	LCD D
Multiplexed buttons,				
Ping Trigger	A3	Atmega	12	LCD DC
LCD reset	A4		13	LCD CLK
Ping Echo	A5		5,6	Motor L
Microphone	A6		7,8	Motor R
Battery Voltage	A7		9,10	Gripper
			11	Speaker

Schematics



Instruction Set

Tab	Section	Name	Detail
control	basic	start	start running a stack of blocks
		repeat	run the inner blocks forever
		repeat <number> times	run the inner blocks a number of times 1 parameter: NUMBER: number of times to loop
		repeat <index variable> from <start> to <end> step <step>	run the inner blocks with an index variable 4 parameters: INDEX VARIABLE: variable name whose value ranges from the start to the end values START: the value with which the loop will start END: the value with which the loop will end STEP: the amount by which the index variable will change after each cycle
		repeat with <variable list> as <values>	run the inner blocks with variables whose values changes over time 2 parameters: VARIABLE LIST: variable names whose values change over time VALUES: values the variables will take on
		repeat while <condition>	run the inner blocks while the condition is true 1 parameter: CONDITION: condition that needs to be true to run the inner stacks
		repeat until <condition>	run the inner blocks until the condition is true 1 parameter: CONDITION: condition that needs to be true to keep running the inner blocks
		wait <duration>	pause for a while 1 parameter: DURATION: time in milliseconds
		if <condition> else	runs the inner blocks if the condition is true 1 parameter: CONDITION: when this condition is true, the inner blocks will run
		switch <expression>	runs the inner block whose value matches the condition 1 parameter: EXPRESSION: expression to switch on *MATCH*: when this matches the expression, the inner blocks will run *Right click on block to access Properties for custom parameters
		<comment>	document your code with text, a schematic or an image 1 parameter: COMMENT: comment which explains your code
	advanced	<first value> <comparer> <comment>	condition block for if and repeat blocks 3 parameters: FIRST VALUE: COMPARER: compare type COMMENT:
		break out of repeat	break out of repeat loop and continue with the following block
		continue to start of repeat	continue to start of repeat loop
		stop	stop the program
		constant:<constants>	assign names to numbers that won't change during the program 1 parameter: CONSTANTS: comma separated list of names and their values
		<code>	inline programming code written in the device's language 1 parameter: CODE: textual representation of code to inline into the program
	state	when in state <state name>	run a set of blocks when the state machine matches this state 1 parameter: STATE NAME: name of the state assigned to this stack of blocks
		run state machine <name of state machine>	run a state machine that manage program behavior using a state variable 1 parameter: NAME OF STATE MACHINE: variable to use for storing machine's state
		set state to <state name>	set the state of the state machine 1 parameter: STATE NAME: the state machine's new state
	event	on <event>	run a set of blocks when something happens 1 parameter: EVENT: condition which starts this stack of blocks
		when <condition>	run a set of blocks when a condition is true 1 parameter: CONDITION: condition which starts this stack of blocks

		task <task name>	group a set of blocks into a named task 1 parameter: TASK NAME: name of task associated with this stack of blocks
		start task <task name>	start a named task and continue right away 1 parameter: TASK NAME: task to run
		start task <task name> and wait	run a named task and wait for it to finish before continuing 1 parameter: TASK NAME: task to run
graphics		print <string> to (<x>,<y>)	print text to a location on the screen 3 parameters: STRING: text to print X: x position of cursor Y: y position of cursor
		draw background <background>	print text to a location on the screen 1 parameter: BACKGROUND: text to print
		draw <sprite> to (<x>,<y>)	print text to a location on the screen 3 parameters: SPRITE: text to print X: x position of cursor Y: y position of cursor
sound		play synthesized song <sound>	play a synthesizer file 1 parameter: SOUND: name of hmus file to play
		stop playing song/effect	stop playing sounds on the synthesizer
		play sound effect <sound>	play a sound effect file 1 parameter: SOUND: name of hsfx file to play
		play score <score>	play a score of music notes as tones 1 parameter: SCORE: score to play
		pluck score <score>	play a score of music notes as plucked notes 1 parameter: SCORE: score to play
		set pluck volume to <volume> tempo to <tempo> sustain to <sustain>	set the parameters of how plucked music sounds 3 parameters: VOLUME: volume to pluck at TEMPO: tempo of plucking SUSTAIN: sustain of pluck
		play wav file <sound> at volume <volume>	play a wav file 2 parameters: SOUND: name of wav file to play VOLUME: volume to play file at
		record sound for <duration> ms	record a sound to memory 1 parameter: DURATION: milliseconds to record sound
		play recorded sound at volume <volume>	play back recorded sound from memory 1 parameter: VOLUME: volume to play sound at
		read microphone	sense the sound level with the microphone
		read microphone amplitude	sense the overall amplitude with the microphone
		read microphone frequency	sense the dominant frequency of sound with the microphone
		play tone <frequency> for <duration> ms	play a tone of music 2 parameters: FREQUENCY: frequency of tone DURATION: milliseconds to play tone
		set tone volume to <volume>	set how loud tones will be played 1 parameter: VOLUME: volume to play tone at
		speak <speech>	speak text using a speech synthesizer 1 parameter: SPEECH: text to say
		speak file <speech>	speak sounds specified in a file 1 parameter: SPEECH: text to say
		spell <speech>	spell text with a speech synthesizer 1 parameter: SPEECH: text to say
		set speech volume to <volume>	set how loud speech will be spoken 1 parameter:

			VOLUME: volume to use for speech
		speech parameters <glottal pitch>,<vibrato pitch>,<vibrato rate>,<pace>	set the parameters of the speech synthesizer 4 parameters: 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 2 parameters: SPEAKER: BASE:
motion	move	move <amount> mm <turn-ratio>	move a distance with a turning ratio 2 parameters: AMOUNT: amount to drive robot in mm, negative to reverse TURN-RATIO: ratio of left and right motor speeds that affects how the robot will turn while moving
		turn <degrees> deg <direction>	turn an amount in a direction 2 parameters: DEGREES: degree to turn robot, negative to reverse DIRECTION: which way to turn the robot, negative to turn left
		move <amount> mm <turn-ratio> in <time> msec	move a distance with a turning ratio in a time 3 parameters: AMOUNT: amount to drive robot in mm, negative to reverse TURN-RATIO: ratio of left and right motor speeds that affects how the robot will turn while moving TIME: milliseconds
		turn <degrees> deg <direction> in <time> msec	turn an amount in a direction in a time 3 parameters: DEGREES: degree to turn robot, negative to reverse DIRECTION: which way to turn the robot, negative to turn left TIME: milliseconds
		set left motor speed to <left> and right to <right>	set the speed of the drive motors; the motors will maintain this speed until the next move block is run 2 parameters: LEFT: speed for the left motor RIGHT: speed for the right motor
		set finish move mode to <action>	set what should happen when move and turn blocks finish running 1 parameter: ACTION:
		set cruise speed <speed at which robot will turn/move>	set the speed of the move and turn blocks 1 parameter: SPEED AT WHICH ROBOT WILL TURN/MOVE:
		configure move with <move>,<turn>	set scale factor for robot move/turn blocks 2 parameters: MOVE: amount robot will move, higher numbers move further TURN: amount robot will turn, higher numbers turn further
		configure motor with <zerol>,<zeror>,<gainl>,<gainr>,<gainl>,<gainr>	set motor offset and scale parameters 6 parameters: ZEROL: zero offset for left motor ZEROR: zero offset for left motor GAINL: +gain for left motor GAINR: +gain for right motor GAINL: -gain for left motor GAINR: -gain for right motor
	servos	set servo <pin> to <position>	set a servo's position 2 parameters: PIN: pin number of servo POSITION: target position for servo
		move servo <pin> to <position>% over <time> msec	ramp a servo's position over time 3 parameters: PIN: pin number of servo POSITION: target position for servo, from -30 to 130 TIME: milliseconds to ramp the servo to the new position
		ramp servo <pin> to <position>% over <time>	ramp a servo's position over time while running other blocks 3 parameters: PIN: 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 <pin>	idle a servo- don't power it 1 parameter: PIN: pin number of servo
		set gripper <shoulder>,<hand> in <time> msec	set gripper 3 parameters:

			SHOULDER: position HAND: position TIME: position
sense		IR distance on <pin>	sense distance with an IR/LED 1 parameter: PIN: pin number of the IR/LED
		ultrasound distance on <port>	sense distance with ultrasound sensor 1 parameter: PORT: port of the sensor
		brightness on <pin>	sense light level with QTI sensor 1 parameter: PIN: pin number of the QTI sensor
		read encoder <side>	read number of encoder pulses 1 parameter: SIDE: side to measure
		read speed <side>	read speed of encoder pulses 1 parameter: SIDE: side to measure
		read line <shoulder>	set gripper 1 parameter: SHOULDER: position
		read ping	set gripper
		read button	set gripper
		read battery	set gripper
		read mic	set gripper
	mouse	mouseX	read the x position of the mouse
		mouseY	read the y position of the mouse
		mouseZ	read the z position of the mouse
		set mouse to (0,0,0)	reset x,y and z
		mouse button down	determine if the mouse button was clicked
	key	key <key> pressed	determine if a keyboard key is pressed 1 parameter: KEY: key to test
	8031	read adc on pin <data out pin>	sense the adc's value 1 parameter: DATA OUT PIN: data out pin followed sequentially by the clock pin and the chip select pin.
		read 10 bit adc on pin <adc>	sense the adc's 10 bit value 1 parameter: ADC: adc pin
	time	reset timer	reset the internal timer
		elapsed time	sense how much time has passed since the last reset
vars	variables	set <variable> to <value>	set a variable to a value 2 parameters: VARIABLE: variable to set VALUE: new value for variable
		change <variable> by <amount>	change a variable's value 2 parameters: VARIABLE: variable to change AMOUNT: amount added to variable
		set bit <bit> of <variable> to <new bit value>	set a bit 3 parameters: BIT: bit to set VARIABLE: variable to modify NEW BIT VALUE: bit value
		get bit <bit> of <variable>	get a bit 2 parameters: BIT: bit to get VARIABLE: variable to inspect
		random(<max>)	return a random number between 0 and the specified maximum 1 parameter: MAX:
		set <variable> to address of <value>	get pointer to a variable 2 parameters: VARIABLE: pointer VALUE: pointee
	arrays	set <array>[<index>] to <value>	set an array's item to a value 3 parameters: ARRAY: array to set INDEX: index of array item to set

			VALUE: new value for array item
		change <array>[<index>] by <amount>	change an array's item 3 parameters: ARRAY: array to change INDEX: index of array item to change AMOUNT: amount added to array item
		get <array>[<index>]	get an array's item 2 parameters: ARRAY: array to retrieve INDEX: index of array item to retrieve
	strings	set <string> to <string>	copy text into the string variable 2 parameters: STRING: string to modify STRING: text to assign
		set <string> to value <new value>	copy the value as text into the string variable 2 parameters: STRING: string NEW VALUE: value to add
		string <first> equals <second>	determine if the two strings are equal 2 parameters: FIRST: first text/string to compare SECOND: second text/string to compare
		lowercase <text>	return the lowercase of the string 1 parameter: TEXT: string to modify
		uppercase <text>	return the uppercase of the string 1 parameter: TEXT: string to modify
		capitalize <text>	capitalize the string 1 parameter: TEXT: string to modify
		reverse <text>	reverse the letters of the string 1 parameter: TEXT: string to modify
		make <count> copies of <text>	return a number of copies of the string 2 parameters: COUNT: number of copies to make TEXT: string to modify
		trim <text>	trim the string 1 parameter: TEXT: string to modify
		pad <text> to length <length> with <pad>	pad string with a string 3 parameters: TEXT: string to modify LENGTH: length to pad to PAD: text/string to pad with
		replace <replacee> with <replacer> in <text>	replace text in a string with another text 3 parameters: REPLACEE: old text REPLACER: new text TEXT: string to modify
		join <new text> to <text>	join one string onto another 2 parameters: NEW TEXT: text/string to add TEXT: string join to
		put <item> split of <text to split> into <result>	split text from a string and put a specified item into another string 3 parameters: ITEM: item of split TEXT TO SPLIT: text/string to split RESULT: result string
		join <new byte> to <text>	join a byte to a string 2 parameters: NEW BYTE: byte to add TEXT: string join to
		get character <text>(<index>)	retrieve an indexed character 2 parameters: TEXT: text/string from which to get a character INDEX: index
		copy substring from <text> starting at <start> for <count> to <output>	copy a substring from a string to another string 4 parameters: TEXT: text/string from which to make substring START: starting index COUNT: characters to copy

			OUTPUT: output string
		copy string beginning with <begin> in <text> starting at <start> to <output>	copy a string that begins with a specified text to another string 4 parameters: BEGIN: text/string to find TEXT: text/string to search in START: starting index OUTPUT: output string
		find index of string <string> in <text> starting at <start>	find where a string matches 3 parameters: STRING: string to find TEXT: text/string to search START: starting index
		find first index of <character> in <text> starting at <start>	find where a character matches 3 parameters: CHARACTER: character to find TEXT: text/string to search START: starting index
		find last index of character <character> in <text> starting at <start>	find the last occurance of a character 3 parameters: CHARACTER: character to find TEXT: text/string to search START: starting index
		length of <text>	calculate the length of the string 1 parameter: TEXT: text/string to count
		convert <text> to a number in base <base>	convert the numeric value in a string into an integer 2 parameters: TEXT: text/string to search BASE: use 10 to convert to decimal
interface	general	program info	display info about variables, arrays and imports. Ctrl-click on imported files to open them
		value viewer	display a table listing variables and their values
		graph	display a graph of variable values over time
		logic levels	display a graph of logic levels over time
		pin activity	display a representation of the chip with live IO pin logic levels
		video	display video from device
	terminal	terminal	display a terminal for text input and output
		send text <text>	send text to the terminal 1 parameter: TEXT: text to send
		send value <value>	send a value to the terminal 1 parameter: VALUE: value to send
		send <text> <value>	send text and colored number to the terminal then clear end of line 2 parameters: TEXT: text to send VALUE: value to send
		send <command>	clear to endof line 1 parameter: COMMAND: command to send
		set position <x>,<y>	set the terminal's position 2 parameters: X: x position of cursor Y: y position of cursor
		set color <color>	set terminal text color 1 parameter: COLOR: color of text
		receive text into <string>	receive text from the terminal and store in a string 1 parameter: STRING: string into which to store received text
		receive number	receive a number from the terminal
		receive byte	receive a byte from the terminal
		received data	test if the terminal has sent something
	controls	background	display an image on which other interface blocks can be docked
		textbox	display and change a variable's value with a textbox
		meter	display a variable's value in a meter
		switch	display and change a variable's value as a switch
		joystick	use a joystick to control two variables
		save	save to file

	advanced	bird's eye view <sim>	display a bird's eye view of a robot environment and simulate the robot's movements 1 parameter: SIM: world
		simulator	display a 3d view of a robot environment and simulate the robot's movements
		swarm <config>	swarm $ 1 parameter: CONFIG: swarm configuration
		integrate with skype	attach to the skype application to allow variable values to be changed and monitored remotely
		integrate with xmlrpc	start a server that supports xmlrpc communication to change and monitor variable values
		integrate with ros	start a ros node to support monitoring and changing variables using the robot operation system
		find fiducial marker <function>	communicate with a fiducial image server to recognize where objects are with vision 1 parameter: FUNCTION:
functions		<name> (<arguments>) locals:<local variables>	group a set of blocks to a named function, you can specify arguments passed into the function as well as local variables 3 parameters: NAME: name for this function ARGUMENTS: arguments passed to this function LOCAL VARIABLES: names of variables that are only used in this function
		return <return value>	return a value from a function 1 parameter: RETURN VALUE: value to return
		synchronize functions <synchronize>	create a set of blocks that will run functions at exact times 1 parameter: SYNCHRONIZE:
		at <time> do <task name>(<task length>)	at a set time, run a named task 3 parameters: TIME: time in msec TASK NAME: task to run TASK LENGTH: length of task
pins	in	count edges on pin <pin> for <duration>	count the number of rising edges on a pin 2 parameters: 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 2 parameters: 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 on a pin 2 parameters: PIN: pin number to measure pulse on STATE: state of pin
		measure pulse on pin <pin> at state <state> with length <state>	measure the duration of a pulse on a pin 3 parameters: PIN: pin number to measure pulse on STATE: state of pin STATE: maximum pulse length in usec
		pin <pin>	read the state of an IO pin 1 parameter: PIN: pin on which state is measured
		duration of discharge on pin <pin>	measure the time until a pin's state changes 1 parameter: PIN: pin which is tested
		shift data in from pin <pin> mode <mode>	shift in 8 bits of data using 3 pins 2 parameters: PIN: pin on which data is shifted in, followed sequentially by the clock pin and then the chip select pin MODE: mode: 0 = MSBPRE 0 Data is msb-first; sample bits before clock pulse 1 = LSBPRE 1 Data is lsb-first; sample bits before clock pulse 2 = MSBPOST 2 Data is msb-first; sample bits after clock pulse 3 = LSBPOST 3 Data is lsb-first; sample bits after clock pulse
		serial in from pin <pin> mode: (<baud>,<mode>,<bits>)	read data using the serial protocol 4 parameters: 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 2 parameters: PIN: pin to transmit on ACKBIT: acknowledge bit
		read <bytes> bytes from <address> into <data>	read data from an i2c eeprom 3 parameters: BYTES: bytes to write ADDRESS: eeprom address DATA: data array
		read adc on pin <pin>	sense the adc's value 1 parameter: PIN: pin to measure
	out	output frequency <frequency> on pin <pin>	continually output a frequency on a pin 2 parameters: 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 3 parameters: FREQUENCY: frequency to output in Hz PIN: pin for output DURATION: milliseconds for output
		set pin <pin> high	set a pin high 1 parameter: PIN: pin to set high
		set pin <pin> low	set a pin low 1 parameter: PIN: pin to set low
		toggle pin <pin>	change a pin's state from high to low/low to high 1 parameter: PIN: pin to change
		configure pin <pin> as <mode>	set a pin's mode 2 parameters: PIN: pin to change MODE: pin
		output pulse length <duration>uSec on pin <pin>	output a pulse 2 parameters: DURATION: microseconds to output pulse PIN: pin to output
		output pwm <duty> on pin <pin>	output a pulse width modulated signal 2 parameters: 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 3 parameters: 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 3 parameters: 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 5 parameters: 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 1 parameter: PIN: i2c scl pin
		send start i2c token on <pin>	send a start i2c token 1 parameter: PIN: i2c scl pin
		write i2c data <data> to pin <pin>	write data with the i2c protocol 2 parameters: DATA: value to transmit PIN: pin to transmit on
		send stop i2c token on <pin>	send a stop i2c token 1 parameter: PIN: i2c scl pin
		write <bytes> bytes of <data> to	write data to an i2c eeprom

🟩		<address>	3 parameters: BYTES: bytes to write DATA: data array ADDRESS: eeprom address
🟩		share array: <arrays>	share arrays with 12Blocks to upload/download data 1 parameter: ARRAYS: type a quoted, commas separated list of array names
🟩		quickly sample the IO pins	quickly sample the IO pins
🟩		analog out <duty> to pin <pin>	output a pulse width modulated signal 2 parameters: DUTY: duty cycle, from 0 to 256 PIN: pin to output

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 set to “On”
- Write or load a program in 12Blocks with the Arduino.TBot library
- Run a program by pressing "Run" in 12Blocks- keep cable connected
- 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.