COIL: The Create Open Interface Library

Installation and Usage Guide

Author: Jesse DeGuire

Certain words and phrases in this document are in color and are links that can be clicked on when reading this document in almost any PDF viwer. Blue links, like in the Table of Contents below, will take you to another part of this document. Cyan links will open up your default web browser and take you to a web page.

Contents

1 Introduction

Welcome! Thank you for downloading and trying out COIL, the Create Open Interface Library. This section will explain what the Create is, how it is controlled, and how this library can help. Start here to get a brief overview of the entire system before you go diving into it.

1.1 The iRobot Create

The Create is a robot that was released by the iRobot Corporation in 2007 in response to users hacking on its popular Roomba vacuum robot. The Create is designed as a hobby robot and is meant to allow its users to program it and get started in the world of robotics. The Create does away with the Roomba's vacuum and replaces it with a cargo bay and a 25-pin expansion port, allowing you to add your own extensions to the robot, such as servos, IR blasters, and even tiny computers. The Create includes 10 built-in demos for you to play with and 42 sensors for you to monitor its operation. For more information, go to iRobot's Create webpage.

1.2 The Create Open Interface

The Create accepts commands through its serial port, located right above the charging jack. Commands can be sent to it from a computer using almost any serial-writing program. These commands consist of one-byte (range: 0-255) commands, called "opcodes", which can have additional one-byte parameters passed to the robot with them. This is a very simple interface and the range of commands you can send to the Create allows for a great deal of freedom in manipulaing the robot. However, the Create can only memorize up to 100 bytes worth of commands in a script, meaning it cannot really do much on its own. Also, sending commands in this interface is difficult and tedious. iRobot sells an accessory called the Command Module which allows for more advanced programming, but it costs more money and is not powerful enough to do advanced functions such as process images from a webcam.

For more infomation on the Create OI, download the Open Interface Specification from iRobot's download page.

1.3 COIL: Create Open Interface Library

This library is a C wrapper for the Open Interface opcodes provided with the Create. What that means is that the user calls a C function to make the robot do something. That C function internally sends the Create the proper opcodes to make that happen. Additional functionality not found in the Create OI is included as well as a convenience to the user. This makes programming against the Create much easier than having to send opcodes on your own, while allowing the user to perform more advanced tasks and leverage the many C libraries availble.

COIL can be run on any POSIX-compliant operating system, such as Linux and Mac OSX. It is possible to connect a small laptop or even smaller computers such as the Gumstix series to the Create to make it a fully-functioning system. A Windows verion of the library is not yet avaible, but is planned.

2 Installing COIL

Installing COIL is just like compiling and installing many applications for Linux and OSX and is easy to do. All you need are the build tools for your operating system, which you already have if you have built programs on your computer before. For Linux, you will need to download packages for GCC and Make (if you have Ubuntu, just get the `build-essential` package). For OSX, just load the Apple Developer Tools from either your OSX installation media or from Apple's developer site.

Open up a console or Terminal and navigate to where you put the files for this library (one directory above where you found this manual). Run **make** and then (as root) **make install** to compile the library and install it to */usr/local/lib*.

Note: OpenCV and the CreateOI library are installed in */usr/local* by default, but some Linux distributions do not look in there. To tell it to do so, create a new file in */etc/ld.so.conf.d/*. It should not matter what you call the file, just end it with *.conf*. In it, put the path you need the distro to look in–in our case `/usr/local/lib`–and save the file. Now execute `ldconfig -v` as root in a terminal to make that path active.

To uninstall the library, run **make uninstall** as root.

3 Compiling and Running Programs

If you want to use COIL functions in your program, just include the header *createoi.h*.

Compiling a program that uses COIL can be done using GCC. Open up a console or Terminal and navigate to where your program is located. Tell GCC to link against COIL with the command line option `-lcreateoi`. For example, if your program is in the file *foo.c*, then you would compile it with the follwing command:

```
gcc -o foo foo.c -lcreateoi
```

Connect your computer to the Create and turn it on. You can then run your program like normal.

4 CreateOI Function Reference

This section describes the functions and structures used in the CreateOI library. If you've looked at the Create Open Interface Specification, you will have a good idea of what many of these commands are capable of. Also available are some additional commands which are not found in the OI. These commands are formed as combinations of basic OI commands and are provided as a convenience to the user.

4.1 Function Documentation

This is a list of the functions available for use with the Create, along with descriptions of what they do. Some functions require special types as one of the parameters. See the next section for more information on those types.

- **int** `startOI (`**char** `* serial)`
  Opens the specified serial port to the Create and starts the Open Interface. The Create will be started in Passive Mode. This command must be called first before any others.
  Parameters:
  serial The location of the serial port connected to the Create

  Returns: 0 on success or -1 on error

- **int** `setBaud (`**oi_baud** `rate)`
  Sets the baud rate between the Create and the computer and waits for 100ms after changing (in compliance with the OI specification). The default is 56kbps.
  Note: At a baud rate of 115200bps, there must be a 20μs gap between each byte or else data loss will occur.
  Parameters:
  rate New baud rate. Must be one of **oi_baud**

  Returns: 0 on success or -1 on error

- **int** `enterPassiveMode ()`
  Puts the Create into Passive Mode. This mode allows reading sensor data, running demos, and changing mode. All other commands are ignored.
  Returns: 0 on success or -1 on error

- **int** `enterSafeMode ()`
  Puts the Create into Safe Mode, allowing control over the Create. The Create will stop and revert to Passive Mode if the cliff or wheel drop sensors are activated or if the charger is plugged in and powered.
  Returns: 0 on success or -1 on error

- **int** `enterFullMode ()`
  Puts the Create into Full Mode, allowing complete control over the Create. The Create's safety features will be turned off in this mode, putting the responsibility of keeping the Create from falling in the programmer's hands.
  Returns: 0 on success or -1 on error

- **int** `runDemo (`**oi_demo** `demo)`
  Starts one of the Create's demo routines. Information about each demo is available in the iRobot Create instruction manual. This will put the Create into Passive Mode.
  Parameters:
  demo The demo to run (or `DEMO_ABORT` to stop the current demo)

  Returns: 0 on success or -1 on error

- **int** `runCoverDemo ()`
  Runs the Cover demo.
  Returns: 0 on success or -1 on error

- **int** `runCoverAndDockDemo ()`
  Runs the Cover and Dock demo.
  Returns: 0 on success or -1 on error

- **int** `runSpotDemo ()`
  Runs the Spot demo.
  Returns: 0 on success or -1 on error

- **int** `drive (`**short** `vel,` **short** `rad)`
  Drives the Create with the given velocity (in mm/s) and turning radius (mm). The velocity ranges from -500 to 500mm/s, with negative velocities driving the Create backward.
  The radius ranges from from -2000 to 2000mm, with positive radii turning the Create left and negative radii turning it right.
  A radius of -1 spins the Create in place clockwise and 1 spins it counter-clockwise. A radius of 0 will make it drive straight.
  Parameters:
  - **vel** The velocity, in mm/s, of the robot
  - **rad** The turning radius, in mm, from the center of the turning circle to the center of the Create

  Returns: 0 on success or -1 on error

- **int** `directDrive (`**short** `Lwheel,` **short** `Rwheel)`
  Controls the velocity of each drive wheel independently. Velocities range from -500 to 500mm/s, with negative values driving that wheel backward.
  Parameters:
  - **Lwheel** The velocity of the left wheel
  - **Rwheel** The velocity of the right wheel

  Returns: 0 on success or -1 on error

- **int** `driveDistance (`**short** `vel,` **short** `rad,` **int** `dist,` **int** `interrupt)`
  Moves the Create at the specified velocity and turning radius until the specified distance is reached, at which point it will stop. If `interrupt` is non-zero, the Create will stop if it detects a possible collision (bump, cliff, overcurrent, or wheel drop sensors are activated).
  Velocity ranges from -500 to 500mm/s (negative values move backward) and turning radius ranges from -2000 to 2000mm (negative values turn right; positive values turn left). A radius of -1 will spin the Create in place clockwise and a radius of 1 will spin it counter-clockwise. A radius of 0 will drive straight.
  Be careful to check your inputs so that the Create does not get stuck in this function. For example, make sure you do not pass a negative distance and a positve velocity.
  Parameters:
  - **vel** Desired velocity in mm/s
  - **rad** Desired turning radius in mm
  - **dist** Distance in mm the Create should travel before stopping
  - **interrupt** Set to 0 to have the Create ignore collisions or non-zero to have it stop on collision

  Returns: Distance travelled or `INT_MIN` on error

- **int** `turn (`**short** `vel,` **short** `rad,` **int** `angle,` **int** `interrupt)`
  Moves the Create at the specified velocity and turning radius until the specified angle is reached, at which point it will stop. If `interrupt` is non-zero, the Create will stop if it detects a possible collision (bump, cliff, overcurrent, or wheel drop sensors are activated).
  Velocity ranges from -500 to 500mm/s (negative values move backward) and turning radius ranges from -2000 to 2000mm (negative values turn right; positive values turn left). A radius of -1 will spin the Create in place clockwise and a radius of 1 will spin it counter-clockwise. A radius of 0 will drive straight.
  Be careful to check your inputs so that the Create does not get stuck in this function. For example, make sure you do not tell the Create to drive straight with this function.
  Parameters:
  - **vel** Desired velocity in mm/s
  - **rad** Desired turning radius in mm
  - **angle** Angle in degrees the Create should turn before stopping

interrupt Set to 0 to have the Create ignore collisions or non-zero to have it stop on collision

Returns: Angle turned or `INT_MIN` on error

- **int** `setLEDState` (**oi_led** `lflags`, **byte** `pColor`, **byte** `pInten`)
  Controls the state of the LEDs on top of the Create. The Play and Advance LEDs are green and can either on or off. Control these by using the **oi_led** flags and the logical-OR operator (`|`). The Power LED is set with two bytes: one for the LEDs color and the other for the light's intensity (higher values are brighter). The color ranges from green (0) to red (255).
  For example, to turn on both LEDs pass `LED_ADVANCE | LED_PLAY` as the first parameter of this function. To turn them off, just pass 0 as the first paramter.
  Parameters:
  lflags LED flags for setting the Play and Advance LEDs
  pColor Color of the Power LED
  pInten Intesity of the Power LED

  Returns: 0 on success or -1 on error

- **int** `setDigitalOuts` (**oi_output** `oflags`)
  Controls the state of the three digital outputs in the 25-pin expansion port. Each output is able to provide up to $20mA$ of current. The state is set using bit flags. For example, to turn on ports 0 and 2, pass `OUTPUT_0 | OUTPUT_2` to this function. Pass 0 to turn them all off. The outputs provide $5V$ when active and $0V$ when inactive.
  Parameters:
  oflags Output flags for setting the state of the digital outputs

  Returns: 0 on success or -1 on error

- **int** `setPWMLowSideDrivers` (**byte** `pwm0`, **byte** `pwm1`, **byte** `pwm2`)
  Specifies the PWM (Pulse Width Modulation) duty cycle for each of the three low side drivers in the Create, with a maximum value of 128. For example, sending a value of 64 would control a driver with with 50% of battery voltage since $128/64 = 0.5$.
  Low side drivers 0 and 1 can provide up to $0.5A$ of current wihle driver 2 can provide up to $1.5A$. If too much current is drawn, the current will be limited and the overcurrent flag in sensor packet 14 (overcurrent sensor) will be set.
  Parameters:
  pwm0 duty cycle for driver 0
  pwm1 duty cycle for driver 1
  pwm2 duty cycle for driver 2

  Returns: 0 on success or -1 on error

- **int** `setLowSideDrivers` (**oi_output** `oflags`)
  Controls the state of the three low side drivers using bit flags. For example, to turn on low side drivers 0 and 1, pass `OUTPUT_0 | OUTPUT_1` to this function. This command activates drivers at 100% PWM duty cycle. Low side drivers 0 and 1 can provide up to $0.5A$ of current wihle driver 2 can provide up to $1.5A$. If too much current is drawn, the current will be limited and the overcurrent flag in sensor packet 14 (overcurrent sensor) will be set.
  Parameters:
  oflags Output flags for setting the state of the three drivers

  Returns: 0 on success or -1 on error

- **int** `sendIRbyte` (**byte** `irbyte`)
  Sends the given IR byte out of low side driver 1, using the format expected by the Create's IR sensor. The Create documentation suggests using a $100ohm$ resistor in parallel with an IR LED and its resistor.
  Parameters:
  irbyte The byte value to send

  Returns: 0 on success or -1 on error

- **int** `writeSong (`**byte** `number,` **byte** `length,` **byte** `* song)`
  Writes and stores a small song into the Create's internal memory. You can store up to 16 songs, with up to 16 notes per song. Each note is associated with a sound from the Create's internal MIDI sequencer, with 31 being the lowest pitch and 127 being the highest. Any value outside that range is considered to be silence. See the iRobot OI Specification for more info.
  The first argument is the song number (0-15). The second argument is the number of notes in the song (1-16). The third arugment is a byte array representing the song. The even entries (0, 2, 4, etc.) are the notes to play and the odd entries (1, 3, 5, etc.) are the durations of these notes in increments of 1/64 of a second (so a value of 32 would play the note for half a second). The size of this array should be equal to twice the second argument.
  Parameters:
  number Song number being stored (0-15)
  length Number of notes in the song (1-16)
  song Byte array containing notes and durations for the song

  Returns: 0 on success or -1 on error

- **int** `playSong (`**byte** `number)`
  Plays a song previously stored in the Create's memory. This command will not do anything if a song is already playing. Check the Song Playing sensor packet to determine if the Create is ready to play a song. This command will return 0 if it was able to request a song from the Create, regardless of whether or not it plays.
  Parameters:
  number Song number to play (0-15)

  Returns: 0 on success or -1 on error

- **int** `readRawSensor (`**oi_sensor** `packet,` **byte** `* buffer,` **int** `size)`
  Requests the Create to return a packet of sensor data which will be put, unformatted, into the specified buffer. Different sensors have different packet sizes, so it is up to you to make sure your buffer is the proper size.
  It is recommended that `readSensor` be used instead, as it will return properly formatted sensor data.
  Parameters:
  packet Sensor packet to read
  buffer Buffer to read raw packets into
  size Number of bytes to read into buffer

  Returns: Number of bytes read or -1 on error

- **int** `readSensor (`**oi_sensor** `packet)`
  Returns data from the specified sensor in a user-readable format. The function will not work for sensor groups, only single sensors. Parameter can be one of **oi_sensor** values.
  Parameters:
  sensor The sensor to get data from

  Returns: Value read from specified sensor or `INT_MIN` on error

- **int** `getDistance ()`
  Returns the distance the Create has travelled since the last time this function was called or the last time the distance sensor was polled.
  Returns: Distance Create travelled since last reading or `INT_MIN` on error

- **int** `getAngle ()`
  Returns the angle the Create has turned since the last time this function was called or the last time the angle sensor was polled.
  Returns: Angle Create turned since last reading or `INT_MIN` on error

- **int** `getVelocity ()`
  Returns the Create's current velocity. Note that this value may differ slightly from the Create's actual velocity due to sensor inaccuracy and wheel slippage.
  Returns: Create's currently requested velocity or `INT_MIN` on error

- **int** `getTurningRadius ()`
  Returns the Create's current turning radius. Note that this value may differ slightly from the Create's actualy turning radius due to sensor inaccuracy and wheel slippage.
  Returns: Create's currently requested turning radius or `INT_MIN` on error

- **int** `getBumpsAndWheelDrops ()`
  Returns the current state of the bumper and wheel drop sensors. The state of each is represented as a single bit in the return value, as shown in the chart below.

Bit	4	3	2	1	0
Decimal	16	8	4	2	1
Sensor	Wheeldrop Caster	Wheeldrop Left	Wheeldrop Right	Bumper Left	Bumper Right

  For example, a return value of 3 means that both the left and right bumper sensors are active since $3 = 2 + 1$.
  Returns: Current state of bumper and wheel drop sensors or `INT_MIN` on error

- **int** `getCliffs ()`
  Returns the current states of the four cliff sensors under the Create's bumper. Each sensor's state is represented as a single bit in the return value, as shown in the chart below.

Bit	3	2	1	0
Decimal	8	4	2	1
Sensor	Left	Front Left	Front Right	Right

  For example, a return value of 12 means that the left and front left cliff sensors are active since $12 = 8 + 4$.
  Returns: Current state of cliff sensors or `INT_MIN` on error

- **int** `* getAllSensors()`
  Returns a pointer to the data from all sensors in the Create. Sensors are in the order given in the Create OI Specification (and in the chart here), starting with Bumps and Wheel Drops (packet 7). Note that packets 15 and 16 are unused and so should always be 0.
  Returns: Pointer to array of sensor data or a NULL pointer on error.

- **int** `readRawSensorList (`**oi_sensor** `* packet_list,` **byte** `num_packets,` **byte** `* buffer,` **int** `size)`
  Requests the Create to return multiple raw sensor data packets. The packets are returned in the order you specify. It is up to you to make sure you are reading the correct number of bytes from the Create.
  Parameters:
  - **packet_list** List of sensor packets to return
  - **num_packets** Number of packets to get
  - **buffer** Buffer to read data into
  - **size** Number of bytes to read into buffer

  Returns: Number of bytes read or -1 on error

- **int** `writeScript (`**byte** `* script,` **byte** `size)`
  Sends the specified script the the Create's internal memory. A script can be up to 100 bytes long.
  Parameters:
  - **script** Script to send to Create
  - **size** Size, in bytes, of the script

  Returns: 0 on success or -1 on error

- **int** `playScript ()`
  Runs the script currently stored in the Create's internal memory. The script remains in memory after it is run, allowing it to be run multiple times.
  Returns: 0 on success or -1 on error

- `byte * getScript()`
  Gets the currently stored script from the Create and returns a pointer to it. The first data byte returned is the number of bytes in the script, followed by the script itself.
  Returns: Pointer to script or a NULL pointer on error

- **double** `waitTime` (**double** `time`)
  Waits the given amount of time, in seconds. The timer has a resolution of 1 microsecond, or 1×10^{-6} seconds; however, accuracy of this function cannot be specified as that depends on the accuracy of the operating system's timer.
  Parameters:
  time Time to wait, in seconds

  Returns: Time passed in as parameter to this function

- **int** `waitDistance` (**int** `dist`, **int** `interrupt`)
  Waits for the Create to travel the given distance (in millimeters). Distance is incremented when the Create travels forward and decremented when it travels backward. If the wheels are passively rotated in either direction, the distance is incremented. The distance sensor will be reset after using this function.
  If `interrupt` is non-zero, this function will exit if the Create detects a possible collision (bump, cliff, overcurrent, or wheel drop sensors are activated).
  The distance travelled is updated once per $20ms$ or 50 times per second.
  Parameters:
  dist Distance to travel in millimeters
  interrupt Set to 0 to ignore collisions or non-zero to have this function exit on collision

  Returns: Distance travelled or `INT_MIN` on error

- **int** `waitAngle` (**int** `angle`, **int** `interrupt`)
  Waits for the Create to turn the given angle (in degrees). The angle is incremented when the Create turns counterclockwise and decremented when it turns clockwise. The angle sensor will be reset after using this function.
  If `interrupt` is non-zero, this function will exit if the Create detects a possible collision (bump, cliff, overcurrent, or wheel drop sensors are activated).
  The angle turned is updated once per $20ms$ or 50 times per second.
  Parameters:
  angle Angle to turn in degrees
  interrupt Set to 0 to ignore collisions or non-zero to have this function exit on collsion

  Returns: Angle turned or `INT_MIN` on error

- **int** `stopOI ()`
  Stops the Create and closes the serial connection to it. The connection can be restarted by calling `startOI`.
  Returns: 0 on success or -1 on error

- **void** `enableDebug ()`
  Turns on debugging, which will print serial transfers to the console window. Reads and writes will be printed in byte-aligned format as they would be transmitted through the serial port.

- **void** `disableDebug ()`
  Turns off debugging so that serial transfers will no longer be printed to the console.

4.2 Type Definitions

These are the different datatypes used for the Create functions.

4.2.1 byte

A byte is the same as an `unsigned char` in C, which has a size of 8 bits. The name "byte" was used as it corresponds to the naming in the Create's documentation. Range is from 0 to 255.

4.2.2 oi_command

Defines constants for all of the opcodes available in the CreateOI. You can use them should you want to write your own functions. The values are in order of opcode.

Type Value	Opcode Name	Opcode Number
OPCODE_START	Start	128
OPCODE_BAUD	Baud	129
OPCODE_SAFE	Safe	131
OPCODE_FULL	Full	132
OPCODE_SPOT	Spot	134
OPCODE_COVER	Cover	135
OPCODE_DEMO	Demo	136
OPCODE_DRIVE	Drive	137
OPCODE_LOW_SIDE_DRIVERS	Low Side Drivers	138
OPCODE_LED	LEDs	139
OPCODE_SONG	Song	140
OPCODE_PLAY_SONG	Play Song	141
OPCODE_SENSORS	Sensors	142
OPCODE_COVER_AND_DOCK	Cover and Dock	143
OPCODE_PWM_LOW_SIDE_DRIVERS	PWM Low Side Drivers	144
OPCODE_DRIVE_DIRECT	Drive Direct	145
OPCODE_DIGITAL_OUTS	Digital Outputs	147
OPCODE_STREAM	Stream	148
OPCODE_QUERY_LIST	Query List	149
OPCODE_PAUSE_RESUME_STREAM	Pause/Resume Stream	150
OPCODE_SEND_IR	Send IR	151
OPCODE_SCRIPT	Script	152
OPCODE_PLAY_SCRIPT	Play Script	153
OPCODE_SHOW_SCRIPT	Show Script	154
OPCODE_WAIT_TIME	Wait Time	155
OPCODE_WAIT_DISTANCE	Wait Distance	156
OPCODE_WAIT_ANGLE	Wait Angle	157
OPCODE_WAIT_EVENT	Wait Event	158

4.2.3 oi_sensor

Defines constants for all 42 sensor packets available in the Create. Not all packets are the same size. Refer to the Create Open Interface documentation for more info. The values are in order of sensor packet number. Packets 16 and 17 are always unused and so are not listed.

Sizes ending in “U” are unsigned (positive values only) and sizes ending in “S” are signed (positive or negative values). It is up to your program to interpret these values correctly. The first six packets are just groups of other packets which may be a mix of signed an unsigned bytes.

Type Value	Packet Name	Packet Size in Bytes
SENSOR_GROUP_0	Group 0 (packets 7-26)	26
SENSOR_GROUP_1	Group 1 (packets 7-16)	10
SENSOR_GROUP_2	Group 2 (packets 17-20)	6
SENSOR_GROUP_3	Group 3 (packets 21-26)	10
SENSOR_GROUP_4	Group 4 (packets 27-34)	14
SENSOR_GROUP_5	Group 5 (packets 35-42)	12
SENSOR_GROUP_ALL	All Sensors	52
SENSOR_BUMPS_AND_WHEEL_DROPS	Bumps and Wheel Drops	1U
SENSOR_WALL	Wall	1U
SENSOR_CLIFF_LEFT	Cliff Left	1U
SENSOR_CLIFF_FRONT_LEFT	Cliff Front Left	1U
SENSOR_CLIFF_FRONT_RIGHT	Cliff Front Right	1U
SENSOR_CLIFF_RIGHT	Cliff Right	1U
SENSOR_VIRTUAL_WALL	Virtual Wall	1U
SENSOR_OVERCURRENT	Overcurrents	1U
SENSOR_INFRARED	Infrared Byte	1U
SENSOR_BUTTONS	Buttons	1U
SENSOR_DISTANCE	Distance	2S
SENSOR_ANGLE	Angle	2S
SENSOR_CHARGING_STATE	Charging State	1U
SENSOR_VOLTAGE	Voltage	2U
SENSOR_CURRENT	Current	2S
SENSOR_BATTERY_TEMP	Battery Temperature	1S
SENSOR_BATTERY_CHARGE	Battery Charge	2U
SENSOR_BATTERY_CAPACITY	Battery Capacity	2U
SENSOR_WALL_SIGNAL	Wall Signal	2U
SENSOR_CLIFF_LEFT_SIGNAL	Cliff Left Signal	2U
SENSOR_CLIFF_FRONT_LEFT_SIGNAL	Cliff Front Left Signal	2U
SENSOR_CLIFF_FRONT_RIGHT_SIGNAL	Cliff Front Right Signal	2U
SENSOR_CLIFF_RIGHT_SIGNAL	Cliff Right Signal	2U
SENSOR_DIGITAL_INPUTS	Cargo Bay Digital Inputs	1U
SENSOR_ANALOG_SIGNAL	Cargo Bay Analog Signal	2U
SENSOR_CHARGING_SOURCES_AVAILABLE	Charging Sources Available	1U
SENSOR_OI_MODE	OI Mode	1U
SENSOR_SONG_NUMBER	Song Number	1U
SENSOR_SONG_IS_PLAYING	Song Playing	1U
SENSOR_NUM_STREAM_PACKETS	Number of Stream Packets	1U
SENSOR_REQUESTED_VELOCITY	Requested Velocity	2S
SENSOR_REQUESTED_RADIUS	Requested Radius	2S
SENSOR_REQUESTED_RIGHT_VEL	Requested Right Velocity	2S
SENSOR_REQUESTED_LEFT_VEL	Requested Left Velocity	2S

4.2.4 oi_baud

Codes to set the buad rate, in bits per second, at which data is sent over the serial port. The default is 56700bps.

Type Value	Speed in Bps
BAUD300	300
BAUD600	600
BAUD1200	1200
BAUD2400	2400
BAUD4800	4800
BAUD9600	9600
BAUD14400	14400
BAUD19200	19200
BAUD28800	28800
BAUD38400	38400
BAUD57600	57600
BAUD115200	115200

4.2.5 oi_demo

Codes for the built-in demos.

Type Value	Demo Name
DEMO_COVER	Cover
DEMO_COVER_AND_DOCK	Cover and Dock
DEMO_SPOT_COVER	Spot Cover
DEMO_MOUSE	Mouse
DEMO_FIGURE_EIGHT	Figure Eight
DEMO_WIMP	Wimp
DEMO_HOME	Home
DEMO_TAG	Tag
DEMO_PACHELBEL	Pachelbel
DEMO_BANJO	Banjo
DEMO_ABORT	Abort current demo

4.2.6 oi_led

Used for turning on and off the LEDs on top of the Create.

Type Value	LED Toggle
LED_ADVANCE	Advance Button LED
LED_PLAY	Play Button LED

4.2.7 oi_output

Used for setting the state of the digital and lowside outputs on the Cargo Bay connector.

Type Value	Output Number
OUTPUT_0	Digital\LSD Output 0
OUTPUT_1	Digital\LSD Output 1
OUTPUT_2	Digital\LSD Output 2

5 Intel OpenCV

OpenCV is an open source computer vision library developed by Intel and released under a BSD license. This library runs under Windows, Linux, and Mac OS X. It is designed for real-time image processing, which is useful for robot vision, but can also be used to process static images and video. OpenCV contains functions for manipulating images and video and for creating basic shapes and graphs. It also includes functions to help with object detection, blob extraction, and some machine learning algorithms. It has a very basic GUI system which can be used for writing simple applications or for prototyping something you are working on.

Check out the OpenCV Wiki to read more about what OpenCV can do and for a complete function reference. It is beyond the scope of this document to go into further detail on how to use OpenCV; however, this section is provided to help you get it installed on your platform if you wish to try it out.

5.1 Installation on Linux

Note: OpenCV and the CreateOI library are installed in */usr/local* by default, but some Linux distributions do not look in there. To tell it to do so, create a new file in */etc/ld.so.conf.d/*. It should not matter what you call the file, just end it with *.conf*. In it, put the path you need the distro to look in–in our case `/usr/local/lib`–and save the file. Now execute `ldconfig -v` as root in a terminal to make that path active.

1. Your webcam and distribution must support Video4Linux2. In Ubuntu, you may need to install the `lipt-plugins-v4l2` package for V4L2 to work properly.

2. You also need to have the UVC (USB Video Class) driver installed. This is a generic driver that allows a wide range of webcams to plug in and "just work", similar to the way many flash drives work. Use your distribution's package manager to find a UVC driver and install it. If that does not work, you can get the source code by entering the following into a console.

   ```
   svn checkout svn://svn.berlios.de/linux-uvc/linux-uvc/trunk linux-uvc
   ```

   Follow the instructions included with the downloaded code to install the driver.

3. Use your distribution's package manager to get the following packages. Unfortunately, you might need to do a bit of searching since not all distributions will necessarily have the same package names. Download the *development* versions of these packages.

   - GTK+ 2.x or higher including headers
   - pkgconfig
   - libpng, zlib, libjpeg, libtiff, libjasper
   - Python 2.3, 2.4, or 2.5
   - libavcodec (this will pull in other codec packages)

4. Download the CVS version of OpenCV by entering the below commands into the console. Press Enter when asked for a password.

   ```
   cvs -d:pserver:anonymous@opencvlibrary.cvs.sourceforge.net:/cvsroot/opencvlibrary
   login

   cvs -z3 -d:pserver:anonymous@opencvlibrary.cvs.sourceforge.net:/cvsroot/opencvlibrary
   co -P opencv
   ```

   This will create a new subdirectory called *opencv*. Change into it.

5. Now you can run **./configure**, **make**, and finally **make install** (run this with root permissions). OpenCV will be installed in */usr/local*. You can run **make check** if you want to make sure OpenCV has installed correctly.

5.2 Installation on OSX

1. Get the latest CVS version of OpenCV by entering the following commands in the Terminal. Press Enter when it asks you for a password.

   ```
   cvs -d:pserver:anonymous@opencvlibrary.cvs.sourceforge.net:/cvsroot/opencvlibrary
   login

   cvs -z3 -d:pserver:anonymous@opencvlibrary.cvs.sourceforge.net:/cvsroot/opencvlibrary
   co -P opencv
   ```

   This will create a new subdirectory called *opencv*. Change into it.

2. You need to configure OpenCV before you can build it. Run **./configure** and let it do its thing.

3. Now you can run **make**, and finally **sudo make install**. OpenCV will be installed in */usr/local*. You can run **make check** if you want to make sure OpenCV has installed correctly.

4. Open up a Terminal and go to your home directory. Run **nano .bashrc** to open or create the file called *.bashrc* for editing. This file is used for user-specific settings for the Terminal. Add the following lines to the file if they are not already there.

   ```
   export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/local/lib/pkgconfig
   export PATH=$PATH:/usr/local/bin
   ```

   Save the file using `Ctrl + O` and exit nano with `Ctrl + X`. You may need to log out and back into OS X for these changes to take effect.

5.3 Compiling and Running a Program

OpenCV may require you to include more than one additional header file in your program to use its functions. See the sample programs included with this library for help.

OpenCV makes use of the pkg-config system for compiling programs that link against it. This is more convenient than having to specify include directories and libraries manually. You can pass the output of the **pkg-config** to GCC using the Bash operator ‘, as shown below.

```
gcc ‘pkg-config --cflags opencv‘ -o foo foo.c ‘pkg-config --libs opencv‘ -lcreateoi
```

Note that the ‘ marks are obtained by pressing the key with the tilde (∼) on it (next to the 1 key on US keyboards).