

# SCR Control Using ROS

## Technical Documentation

### Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Setup</b>	<b>2</b>
2.1	Installing ROS . . . . .	2
2.1.1	Installing ROS on a raspberry pi . . . . .	2
2.2	Creating a Workspace . . . . .	2
2.3	Downloading the Package . . . . .	2
2.4	Running Scripts . . . . .	3
2.5	Running ROS from multiple machines . . . . .	3
2.5.1	Test communication between devices . . . . .	3
2.5.2	Set environment variables . . . . .	3
2.5.3	Run the server and client . . . . .	3
<b>3</b>	<b>Server</b>	<b>4</b>
3.1	Running the Server . . . . .	4
3.2	Closing the Server . . . . .	4
<b>4</b>	<b>Client</b>	<b>4</b>
4.1	Running the Client . . . . .	4
<b>5</b>	<b>Lights</b>	<b>4</b>
5.1	Configuration . . . . .	4
5.2	Commands . . . . .	6
<b>6</b>	<b>Blinds</b>	<b>7</b>
6.1	Configuration . . . . .	7
6.2	Commands . . . . .	8
<b>7</b>	<b>Color Sensors</b>	<b>8</b>
7.1	Configuration . . . . .	9
7.2	Commands . . . . .	9
<b>8</b>	<b>HVAC</b>	<b>9</b>
8.1	Configuration . . . . .	9
8.2	Commands . . . . .	9
<b>9</b>	<b>Time of Flight</b>	<b>10</b>
9.1	Configuration . . . . .	10
9.2	Commands . . . . .	11
<b>10</b>	<b>In Python</b>	<b>11</b>
10.1	Imports . . . . .	11
10.2	Commands . . . . .	11

# 1 Introduction

This project is a ROS package containing scripts to control the lights, blinds and HVAC, and read data from sensors in the Smart Conference Room. In order to run any of the scripts, ROS must be installed and `roscore` must be active. Information on ROS can be found at <http://wiki.ros.org/>.

## 2 Setup

Robot Operating System (ROS) must be installed in order to run these scripts.

### 2.1 Installing ROS

To install ROS Kinetic, follow the tutorial on the ROS website (<http://wiki.ros.org/ROS/Installation>). After installing ROS, you need to source its `setup.*sh` files. To do this, run:

```
$ source /opt/ros/<distro>/setup.bash
```

This command needs to be run on every new shell in order to access ROS commands, unless you add it to your `.bashrc`.

See the tutorial at <http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment> for more information.

To use ROS on windows, you can install Ubuntu using Windows Subsystem for Linux (WSL). For more information, use the following microsoft tutorial: <https://docs.microsoft.com/en-us/windows/wsl/install-win10>

#### 2.1.1 Installing ROS on a raspberry pi

An SD Card Image with Ubuntu 16.04 (LXDE) and ROS Kinetic installed for the Raspberry Pi 3 can be downloaded at <https://downloads.ubiquityrobotics.com/pi.html>

For other raspberry pi models follow the tutorial at <http://wiki.ros.org/ROSBerryPi/Installing%20ROS%20Kinetic%20on%20the%20Raspberry%20Pi>. Note that this may take multiple hours.

### 2.2 Creating a Workspace

Making the ROS package requires a catkin workspace. To create a new workspace, create a new directory and run `catkin_make` in it. For example:

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```

Now you have a workspace, which also needs to be sourced. To source your workspace:

```
$ source devel/setup.bash
```

As the workspace also needs to be sourced in every new shell, adding a command to source the workspace to your `.bashrc` is very convenient.

See the tutorial at <http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment> for more information.

### 2.3 Downloading the Package

Now ROS should be all setup. The package is available from [https://github.com/onetarp/light\\_control](https://github.com/onetarp/light_control). You can download or clone the file into the `src` folder in your catkin workspace.

## 2.4 Running Scripts

Before you start running any scripts from the package two things need to be done.

1. Run `catkin_make` in the top level of your catkin workspace
2. Run `roscore`

`roscore` launches the backbone of any ROS-based-system which is required for ROS nodes to communicate. `roscore` can be closed at any time by pressing `Ctrl` + `Q`.

Now you can run any command in the `light_control` package by opening a new shell and using:

```
$ rosrn scr_control [script_name] <commands>
```

In general, you will need to run the server for any component of the Smart Conference Room before running any of the clients.

## 2.5 Running ROS from multiple machines

In order to run the client and server on separate machines, first install ROS on both devices. Follow instructions in sections 2.1 - 2.4 to install ROS on both machines.

### 2.5.1 Test communication between devices

First, it is important to test whether the machines can communicate with eachother.

To test communication between multiple devices use:

```
$ ping [ip or hostname of other device]
```

To check the ip address of the current device, run `ifconfig`

If one or more devices are running windows, it may be necessary to edit firewall settings. To do so, open Windows Defender Firewall and navigate to **Advanced Settings > Inbound Rules**. Enable rules titled **File and Printer Sharing (Echo Request - ICMPv4-In)**

### 2.5.2 Set environment variables

On the master device (that will be running `roscore`) run `roscore`

Part of the output should read: `ROS_MASTER_URI=http://[device]:[port]/`

On all machines, run the following command using the device and port shown on the master device.

```
$ export ROS_MASTER_URI=http://[device]:[port]
```

Additionally, on each machine, run the following using the device ip or name of that individual machine

```
$ export ROS_IP=[device]
```

To avoid setting these variables in each terminal, these commands can be added to your `.bashrc` (See the bottom of 2.2)

### 2.5.3 Run the server and client

You should now be able to run the server and client scripts on separate machines.

If you are still having troubles, see the relevant ROS tutorial: <http://wiki.ros.org/ROS/Tutorials/MultipleMachines>

## 3 Server

### 3.1 Running the Server

To run any server:

```
roslaunch scr_control SCR_[]server_name]_server.py
```

The names of each client can be found at the beginning of their respective section.

If the server prints:

```
Unable to register with master node [http://localhost:11311]: master may not be running yet.  
Will keep trying.
```

Then `roscore` is not running. Open another shell and run `roscore` and the server should run.

### 3.2 Closing the Server

The server can be closed at any time by pressing `Ctrl+C` in the shell in which the server is running.

## 4 Client

### 4.1 Running the Client

For the client to run any commands, the corresponding server must be running first.

To run the client:

```
$ roslaunch scr_control SCR_[client_name]_client.py [command] [arguments]
```

The names of each client can be found at the beginning of their respective section.

## 5 Lights

The lights in the Smart Conference Room have many parameters controlling their color. Each light has five color options, red, blue, green, amber and white. Each color can be set from 0% to 100% intensity to produce different combinations of colors. Each light can also be set to specific CCT's and overall intensities.

To run the light server run:

```
roslaunch scr_control SCR_OctaLight_server.py
```

To run the light client run:

```
roslaunch scr_control SCR_OctaLight_client.py [command] [arguments]
```

### 5.1 Configuration

Three text files are required to correctly configure the light server.

1. `SCR_OctaLight_conf.txt`

This file contains a list of the IP addresses of the lights in the system, as well as a position array of the lights. The format of the file is as follows:

```

192.168.0.111
192.168.0.112
.
.
.
192.168.0.n
-
10011001
00100100
10011001

```

The first part of the configuration file is a list of IP addresses for every light in the system. The second part is an array representing the position of lights in the room. The number of 1's in the array must equal the number of IP addresses. A 1 represents a light and a 0 represents empty space. The IP addresses are assigned to positions in order. In the above example, light 1 would have IP address 192.168.0.111 and coordinates (0,0). Light 2 would have IP address 192.168.0.111 and coordinates (0,2). Light n would have IP address 192.168.0.120 and coordinates (4,2). The two parts are separated by a dash (-).

## 2. SCR\_OctaLight\_CCT.txt

This file contains the color values needed to produce different correlated color temperatures (CCT's). The CCT's available range from 1800 K to 10000 K in increments of 100. The format of the file is as follows:

```

#cct    red amber green blue white
1800 0.840241787 0.909269192 1 0.011311029 0.194062057
1900 0.763359902 0.859734808 1 0.00379065 0.251035096
.
.
.
10000 0.087561979 0 0.005743162 1 0.481888995

```

The first column is a CCT value, followed by the values for red, amber, green, blue, and white in each following column. Each column is separated by a single tab. The values for each color are percentages of intensity for that individual color with 1 being maximum intensity.

## 3. SCR\_PentaLight\_int.txt

This file contains coefficients for each color of light to achieve a linear curve for light levels ranging from 0% to 100% intensity. The format of the file is as follows:

```

-0.0826 1.0943
-0.1120 1.1145
-0.1451 1.1478
-0.1718 1.1782
-0.0476 1.0513

```

The first value on each line corresponds to the squared term when determining the intensity of a particular color. The second value on each line corresponds to the first power term. The coefficients on the first, second, third, fourth and fifth line correspond to blue, green, amber, red and white respectively. For example, given a value for the intensity of blue  $i_b$  between 0% and 100%, the final intensity for blue  $I_b$  would be calculated as follows:

$$I_b = -.0826i_b^2 + 1.0943i_b$$

## 5.2 Commands

**cct** [x\_coord] [y\_coord] [CCT\_value] [intensity\_percent]

Sets the CCT of the light at the input coordinates to the input CCT value at the input intensity.  
Returns the new state of the specified light.

*x\_coord* the x coordinate of the light to be changed

*y\_coord* the y coordinate of the light to be changed

*CCT\_value* the light temperature to change the specified light to  
integer value between 1800 and 10000

*intensity\_percent* the percent intensity to set the specified light to  
a percentage between 0 and 100

**sources** [x\_coord] [y\_coord] [blue1\_percent] [blue2\_percent] [blue3\_percent]  
[lime\_percent] [amber\_percent] [orange\_percent] [red1\_percent] [red2\_percent]

Sets the color of the light at the input coordinates to the input values.  
Returns the new state of the specified light.

*x\_coord* the x coordinate of the light to be changed

*y\_coord* the y coordinate of the light to be changed

*blue1\_percent* the first percent of blue to set the specified light to  
a percentage between 0 and 100

*blue2\_percent* the second percent of blue to set the specified light to  
a percentage between 0 and 100

*blue3\_percent* the third percent of blue to set the specified light to  
a percentage between 0 and 100

*lime\_percent* the percent lime to set the specified light to  
a percentage between 0 and 100

*amber\_percent* the percent amber to set the specified light to  
a percentage between 0 and 100

*orange\_percent* the percent orange to set the specified light to  
a percentage between 0 and 100

*red1\_percent* the first percent of red to set the specified light to  
integer value between 1800 and 10000

*red2\_percent* the second percent of red to set the specified light to  
integer value between 1800 and 10000

**cct\_all** [CCT\_value] [intensity\_percent]

Sets the CCT of all lights to the input CCT value at the input intensity.  
Returns the new state of the last light changed.

*CCT\_value* the light temperature to change all lights to  
integer value between 1800 and 10000

*intensity\_percent* the percent intensity to set all lights to  
a percentage between 0 and 100

**sources\_all** [blue1\_percent] [blue2\_percent] [blue3\_percent] [lime\_percent]  
[amber\_percent] [orange\_percent] [red1\_percent] [red2\_percent]

Sets the color of all lights at the input coordinates to the input values.  
Returns the new state of the last changed light.

*blue1\_percent* the first percent of blue to set the specified light to a percentage between 0 and 100

*blue2\_percent* the second percent of blue to set the specified light to a percentage between 0 and 100

*blue3\_percent* the third percent of blue to set the specified light to a percentage between 0 and 100

*lime\_percent* the percent lime to set the specified light to a percentage between 0 and 100

*amber\_percent* the percent amber to set the specified light to a percentage between 0 and 100

*orange\_percent* the percent orange to set the specified light to a percentage between 0 and 100

*red1\_percent* the first percent of red to set the specified light to integer value between 1800 and 10000

*red2\_percent* the second percent of red to set the specified light to integer value between 1800 and 10000

**get\_cct** [x\_coord] [y\_coord]

Returns the CCT value of the light at the input coordinates if the CCT value has been changed before. If there is no light at the input coordinates, returns with error.

*x\_coord* the x coordinate of the light from which to get CCT value

*y\_coord* the y coordinate of the light from which to get CCT value

**get\_int** [x\_coord] [y\_coord]

Returns the intensity percent of the light at the input coordinates if the intensity percent has been changed before. If there is no light at the input coordinates, returns with error.

*x\_coord* the x coordinate of the light from which to get intensity percent

*y\_coord* the y coordinate of the light from which to get intensity percent

**get\_lights** Returns a list of all lights as their coordinates [x, y]

**help** prints a list of commands and their arguments

## 6 Blinds

The blinds can be raised or lowered to any position. The slats can also be tilted to any angle.

To run the Blind server run:

```
roslaunch scr_control SCR_Blind_server.py
```

To run the light client run:

```
roslaunch scr_control SCR_Blind_client.py [command] [arguments]
```

### 6.1 Configuration

One text file is required to correctly configure the blind server.

#### 1. SCR\_blind\_conf.txt

This file contains the IP address of the blind controller and a list of each blind's orientation. The format of the file is as follows:

```
192.168.0.130
-
N1
N2
E1
E2
```

The first part is the IP address of the blind controller. The second part is a list of blinds. Each blind has an orientation (N,E,S,W) and a number (1,...,n). For each orientation, the numbering of blinds begins at 1 and increases. The two parts are separated by a dash (-).

## 6.2 Commands

**lift** [blind] [percent]

Lifts the input blind to the input position  
Returns the new position of the blind

*blind* the blind to lift

a string containing orientation + number, ex. N1 or S4 or E3

*percent* the position to lift the blind to

a percentage between 0 and 100

**tilt** [blind] [percent]

Tilts the slats on the input blind to the input position  
Returns the new position of the slats

*blind* the blind to tilt

a string containing orientation + number, ex. N1 or S4 or E3

*percent* the position to tilt the blind to

a percentage between 0 and 100

**lift\_all** [percent]

Lifts all blinds to the input position  
Returns the new position of the blinds

*percent* the position to lift the blinds to

a percentage between 0 and 100

**tilt\_all** [percent]

Tilts the slats on all blinds to the input position  
Returns the new position of the slats

*percent* the position to tilt the blinds to

a percentage between 0 and 100

**get\_blinds** prints a list of all blind names ex. [N1, N2, E1]

**help** prints a list of commands and their arguments

## 7 Color Sensors

The color sensors read the color data of the room.

To run the Color Sensor server run:

```
roslaunch scr_control SCR_ColorSensor_server.py
```

To run the Color Sensor client run:

```
roslaunch scr_control SCR_ColorSensor_client.py [command] [arguments]
```



## 7.1 Configuration

One text file is required to correctly configure the blind server.

1. `SCR_blind_conf.txt`

This file contains the IP address of the sensor controller. The format of the file is as follows:

```
192.168.0.41
```

## 7.2 Commands

**read\_all** Reads data from every sensor in the system  
Returns the data read from every sensor

**read** [*sensor*]  
Reads data from the input sensor Returns data read from the sensor

*sensor* the sensor to read data from  
an integer number, 1, 6, ..., n

**inte.time** [*time*]  
Sets the integration time for the color sensors

*time* the time in milliseconds to set the integration time to  
an integer number between 1 and 250

**help** prints a list of commands and their arguments

## 8 HVAC

The HVAC system

To run the HVAC server run:

```
roslaunch scr_control SCR_HVAC_server.py
```

To run the HVAC client run:

```
roslaunch scr_control SCR_HVAC_client.py [command] [arguments]
```

### 8.1 Configuration

One text file is required to correctly configure the HVAC server.

1. `SCR_HVAC_conf.txt`

This file contains the IP address of the HVAC controller. The format of the file is as follows:

```
192.168.0.36
```

### 8.2 Commands

**set\_temp** [*temp*] sets the temperature of the room

*temp* temperature, in Celsius, to set the room to  
a float

**set\_fansp** [*speed*] sets the fan speed of the room

*speed* speed to set the fan to  
a string, string = off—low—medium—high

**set\_ep** [ep] [val] sets ep for the system

*ep* ep value to set  
an int

*val* the value the ep will be set to  
an int from 0-100

**set\_bms** enables building maintenance system to take control

**get\_temp** gets current temperature reading from each sensor  
returns a list with the temperature readings from each sensor

**get\_ep** gets current ep values from each ep  
returns a list with the ep values for each ep

**get\_co2** gets current CO2 reading in the room  
returns a list with the co2 readings from each sensor

**get\_rh** gets the current relative humidity in the room  
returns a list with the co2 readings from each sensor

**help** prints a list of commands and their arguments

## 9 Time of Flight

To run the ROS TOF server run:

```
roslaunch scr_control SCR_TOF_server.py
```

To run the TOF client run:

```
roslaunch scr_control SCR_TOF_client.py [command] [arguments]
```

A second server must also be running to communicate with the TOF sensors, which is held in a separate git repository [https://github.com/duffy4/tof\\_control](https://github.com/duffy4/tof_control). This second server exists because the TOF sensor API is in c++. To run this second server, run:

```
./tof_control/SCR/tof
```

### 9.1 Configuration

One text file is required to correctly configure the TOF server.

1. `SCR_TOF_conf.txt`

This file contains the IP address of the device running the c++ server. The format of the file is as follows:

```
127.0.0.1
```

## 9.2 Commands

**start\_counting** turns on the TOF sensors and sets them to start gathering data.

**stop\_counting** turns off the TOF sensors.

**get\_distances** [sensor\_num] returns a 2D list with the data from the TOF sensor

*sensor\_num* the sensor that you wish to get data of  
an int

**get\_distances.all** returns a 2D list representing a map of the room using data from the TOF sensors. Areas not covered by the TOF sensors are set to 0.

**help** prints a list of all commands, what they do, and what arguments they take

## 10 In Python

### 10.1 Imports

Because python only searches the current path for imports, you must add the corresponding script folder to your python path before importing any clients.

Additionally, all functions have the optional argument debug which defaults to False, and determines whether debug information should be printed.

### 10.2 Commands

Running any of the commands in a python script is similar to running from the command line. In the case of the lights, to run light commands from a python script use:

```
SCR_PentaLight_client.[command]([arguments])
```

For example:

```
import sys
sys.path.append(\\path/to/package/scr_control/scripts/lights")
import SCR_PentaLight_client as light_control
```

```
light_control.ragbw_all(100, 0, 0, 0, 0, debug = True)
# Prints out: "Changing all lights to red:100% amber:0% green:0% blue:0% white:0%"
```