Installation Guide   16 Feb 2024

The sensorCAM is an advanced concept foreign to most model railroaders and consequently deals with many unfamiliar issues. It also calls on multiple software applications to address them. The initial software setup for an inexperienced tinkerer is consequently involved and needing perseverance. The Arduino IDE, the ESP32 libraries, the Processing 4 App, the EX-Command Station and EX-rail (if used) combine to create a unique system supporting the sensorCAM's own software. Any prior knowledge of these will vastly expedite the installation.

Outline:

 This installation process is divided into 12 steps outlined below. It is recommended that it be treated as a learning exercise and that testing of progress occur after each major step. For example, familiarization with Arduino IDE should be developed before moving to the ESP32 example, and then familiarize with the ESP32- CAM before moving on to the sensorCAM loading.

As the sensorCAM is not for Conductors, and perhaps only advanced Tinkerers, it is anticipated that users will already be familiar with the Arduino IDE and maybe already have toyed with the ESP32-CAM.   We have not therefore included much detail on the first few steps but rather referred to existing documentation elsewhere (e.g. Arduino & youtube). Most detail is provided in steps 4 to 8 at this stage.

The chapters of the full manual may be referred to where additional detail is sought. The 12 steps are:

1.   Check youtube video then acquire ESP32-CAM and MB/FTDI
2.   install Arduino IDE with the ESP32 libraries (includes a WiFi CAM example)
3.   Load the CAM with the WiFi example and test.
4.   Install the sensorCAM.ino software and configure.
5.   Test the basic sensorCAM functions to confirm functional install
6.   Load Processing 4 app. and sensorCAM.pde code
7.   Use the Processing 4 app to replace the Arduino Monitor. Further familiarise and test.
8.   With a fixed camera, create test sensors and test detection with moving targets.
9.   Setup the CAM viewing the model railroad and test a virtual sensor with moving rolling stock.
10.   Optimise parameters for best performance.
11.   Connect CAM to an i2c interface (e.g. PCA9515A or better)
12.   Depending on system, integrate sensorCAM into Command Station using appropriate code.

Step 1. ESP-32 CAM

To understand the animal, watch the video at [ESP32 CAM - 10 Dollar Camera for IoT Projects - YouTube](#)

Be aware that you need a USB adapter and the ESP32-CAM-MB is preferred as it avoids hazards associated with a "wired" FTDI power where errors can easily result in CAM destruction. Programming is also easier. Acquire a CAM from your favourite source. Also be prepared for defective product so a spare CAM may be worthwhile. I have had 2 faulty ov2640 modules.

Step 2. Install the Arduino IDE

If you are not already familiar with this IDE (Integrated Development Environment), rather than go into details that are already covered in great detail on the Arduino web page, just follow the instructions in the following link and then return here.

[Arduino IDE Guide](#)

Step 2A. Install the IDE ESP32 libraries (including for ESP32-CAM).

If you are new to ESP32, the board manager for ESP32 must be loaded. This is explained in the video at the 10minute mark.

[Introduction to ESP32 - Getting Started - YouTube](#)

To get an introduction to the ESP32-CAM review this

[ESP32 CAM - 10 Dollar Camera for IoT Projects - YouTube](#)

Step 3. Load and test the ESP32-CAM example

Select the tools - board manager and the AI thinker ESP32 CAM . Then run the example as per youtube at 9:30 minutes in. Run the ESP32 - camera - webserver example

[ESP32 CAM - 10 Dollar Camera for IoT Projects - YouTube](#)

Note: With the MB board there is a push button that replaces the FTDI programming link. Hold the button in when powering up the CAM to initiate programming mode.
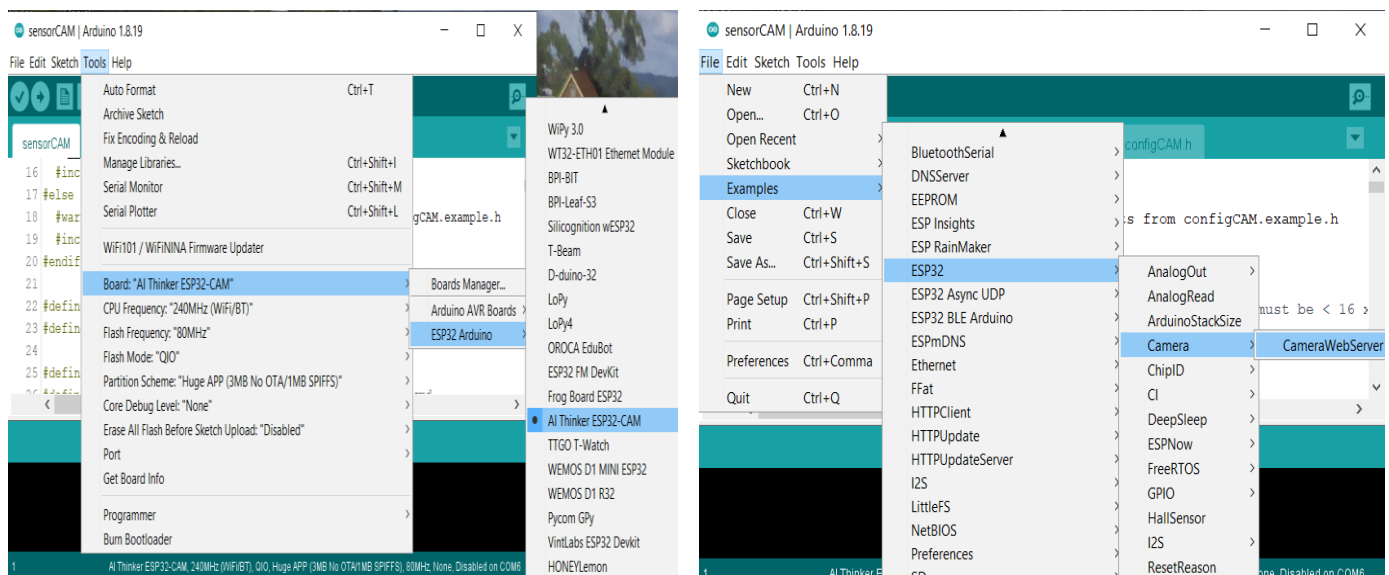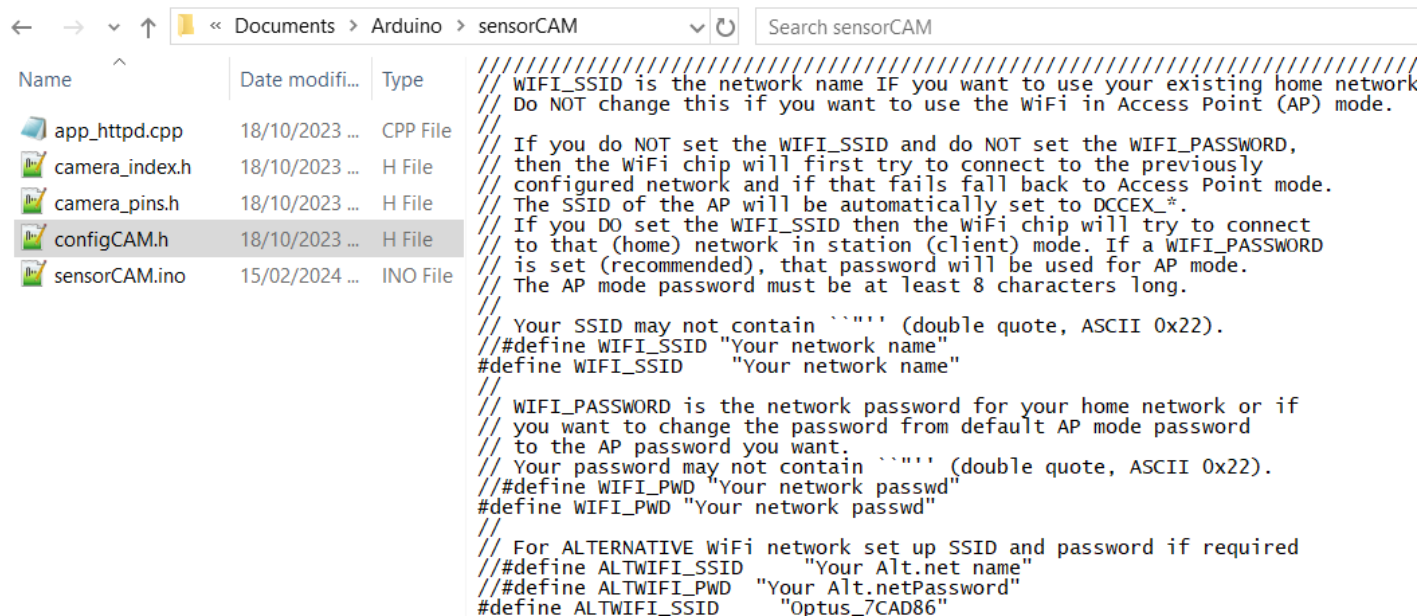
Step4. Install the sensor CAM

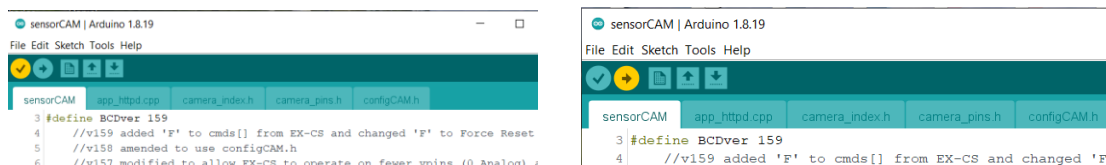Copy (or Clone) the Github software into your sketch directory Arduino/sensorCAM

https://github.com/DCC-EX/EX-SensorCAM

The essential files for now are shown below. Other files may be used in later install steps.

Using the configCAM.example.h as a guide, create and edit a configCAM.h to reflect your WiFi network name and password. Then with the correct com port and board selected, compile the sensorCAM.ino sketch. If all is well, load it into the CAM (replacing the ESP32 WiFi example)

Step 5. Test the basic sensorCAM functions

The sensorCAM has a suite of command codes with which you will need familiarity. Refer to the manual summary Appendix A or <u>Chapter 5</u> (Configuration) for more detailed description. For now, open the Arduino monitor and the initialization procedure should scroll out. After about 15 seconds you can enter the '**w**' (wait) command into the monitor and the scrolling should pause. Another command such as '**ver**' will give relevant command response, or **Enter** alone will resume scrolling tabular output. The sensors still need configuring before they can work.

```
COM6                                                          —    □    ✕

|                                                                  Send

100mS;   T39 N0 R1091 A1057 M2
100mS;   T39 N0 R1091 A1090 M2
100mS;   T39 N0 R1091 A1106 M2
100mS;   T39 N0 R1091 A1115 M2
100mS;   T39 N0 R1091 A1098 M2
100mS;   T39 N0 R1091 A1095 M2
100mS;   T39 N0 R1091 A1109 M2
w
**************
Waiting for new line
ver
**************
CAM software version (BCDver): 159
Waiting for new line

<                                                                 >
☑ Autoscroll ☐ Show timestamp        Newline ∨   115200 baud ∨  Clear output
```

At this time a NEW CAM will have NO sensors set and is a clean slate needing meaningful configuration data. If all is well, this would be a good time to familiarize oneself with some commands starting with **k10,50,80** to DEFINE your first sensor, followed by **i10** to read back the new sensor S10 coordinates (50,80).   Try **a10** to enable sensor 10, quickly followed by **w** to stop scrolling of new sensor values.

```
COM6                                                          —    □    ✕

w                                                                 Send

k10,50,80
**************
(k##,rrr,xxx)setting new coordinates for Sensor[1/0] to r:50 x:80
Waiting for new line
i10
**************
(i##,$$) Info: Sensor 1/0(8) enabled:0 status:0 false/unoccupied  r= 50 x= 80  brightness A787
Waiting for new line
a10
**************
(a##) Activate: sensor 1/0 with fresh reference - bsNo 010
updated Sensor_ref[] HEX bytes(0-5)(BGRBGR): 8 10 a a 10 a
198802mS;  T39 N0 R1091 A787 M2          10:?_113_?* Ref 010
8mS;  T39 N0 R1091 A1091 M2     10:o145##*
100mS;   T39 N0 R1091 A1096 M2    10:o145##*
100mS;   T39 N0 R1091 A1075 M2    10:o145##*
100mS;   T39 N0 R1091 A1043 M2    10:o145##*
100mS;   T39 N0 R1091 A1038 M2    10:o145##*
```
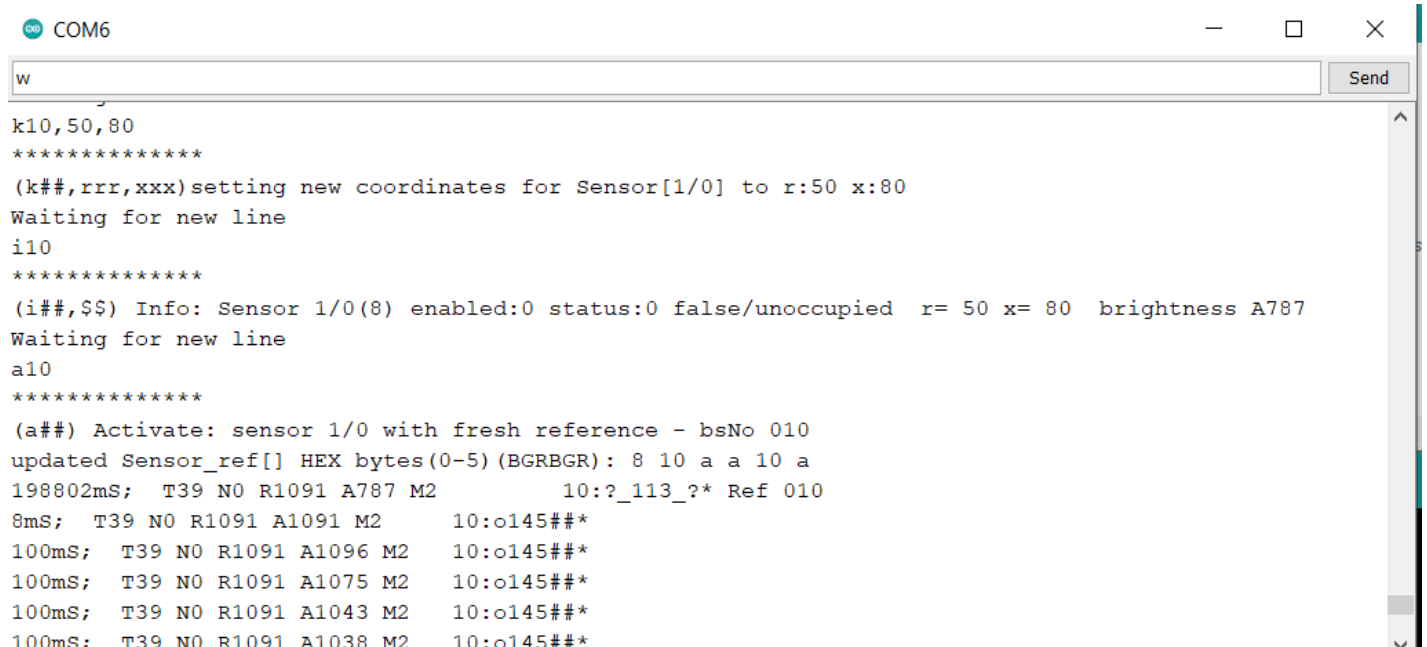
Then try **a11,60,90** to DEFINE *and* ENABLE sensor 11. The '**p1**' pointer command should now list the new sensors and the "difference" score hopefully under 39 for any stable sensor provided the camera is fixed and pointing at a reasonably lit area.

```
                                                              Send
a11,60,90
**************
(a##,rrr,xxx)setting new coordinates for Sensor[1/1] to r:60 x:90
Waiting for new line
(a##) Activate: sensor 1/1 with fresh reference - bsNo 011
updated Sensor_ref[] HEX bytes(0-5)(BGRBGR): e e e e e e
p1
**************
(p##) Position Pointers: TL corners for DEFINED Sensors to block 1
Block 0;
Block 1; s[0]: r= 50 x= 80 s[1]: r= 60 x= 90
Waiting for new line
```

An '**r00**' command will take an internal reference photo. Follow with '**w**'.

```
w                                                             Send
r00
**************
(r##) refresh Reference for sensor[##]. (default r00 for all) (MUST BE UNOCCUPIED!)
No action as Sensor(bsNo) undefined
476723mS;  T39 N0 R1091 A873 M2           10:o145##* 11:oo65##* Ref 011
7mS;  T39 N0 R1091 A1022 M2      10:o145##* 11:o119##*
100mS;  T39 N0 R1091 A1020 M2    10:o145##* 11:o119##*
(r##) refresh Reference for sensor[##]. (default r00 for all) (MUST BE UNOCCUPIED!)
References: enable new Sensor_ref[1/1]; HEX bytes[0-6]: 3E 3F 3E  3E 3F 3E
full new ref[bsNo] from current frame only - average ref still to be calculated
114mS; SUS  T39 N0 R1091 A2992 M2        10:o145##* 11:?_32##* Ref 011
76mS;  T39 N0 R1091 A1047 M2     10:o145##* 11:?_32##*
100mS;  T39 N0 R1091 A1044 M2    10:o145##* 11:--32--*
100mS;  T39 N0 R1091 A1032 M2    10:o145##* 11:--32--*
100mS;  T39 N0 R1091 A1039 M2    10:o145##* 11:--32--*
w
**************
Waiting for new line
```

An **Enter** should resume scrolling with live sensor states and a hand in front of the CAM *may* trip them (depending on lighting etc.). The **w** command will again pause scrolling. To save the new sensor settings in EPROM enter the '**e**' command. You can undefine(erase) a sensor with '**u**', redefine/relocate a sensor with '**a**' inspect/interrogate a sensor with '**i**', even display the digital (hex) values of pixels in sensor frame with '**f**'. Refer to manual chapter 5 for more details on parameters, but do read Section 4.1 (Notation) first.

Note: There are other (easier) ways to position new sensors. The coordinate method is best for "tweaking"positions by small amounts. Step 6 below leads to the preferable visual setting method.

```
 ∞ COM6                                               —    □    ×
                                                              Send
a11,60,85
**************
(a##,rrr,xxx)setting new coordinates for Sensor[1/1] to r:60 x:85
Waiting for new line
(a##) Activate: sensor 1/1 with fresh reference - bsNo 011
updated Sensor_ref[] HEX bytes(0-5)(BGRBGR): 6 b 8 6 b a
i11
**************
(i##,$$) Info: Sensor 1/1(9) enabled:1 status:0 false/unoccupied  r= 60 x= 85  brightness A524
Waiting for new line
f11
**************
(f##) Frame: Print fb ref & newest sample for S011
06 0B 08   06 0B 0A   0A 09 08   0A 09 08    offsetBytes 432 bsNo=1/1 -> 06 0B 08   06 0B 0A   0A 09 08   0A 09 08
08 0C 10   06 0A 0E   08 09 0C   08 0A 0C    offsetBytes 432 bsNo=1/1 -> 08 0C 10   06 0A 0E   08 09 0C   08 0A 0C
08 0E 10   06 0D 0E   08 0C 0E   08 0C 10    offsetBytes 432 bsNo=1/1 -> 08 0E 10   06 0D 0E   08 0C 0E   08 0C 10
08 0E 12   08 0C 10   0A 0C 10   0A 0C 10    offsetBytes 432 bsNo=1/1 -> 08 0E 12   08 0C 10   0A 0C 10   0A 0C 10
Waiting for new line
```

Step 6. Load Processing 4

The Arduino monitor is not "visual". The Processing 4 software enables one to see what the CAM sees and permits one to place sensors visually. Close down the Arduino IDE to free up the sesorCAM USB com port.

Download the "processing-4.3-windows-r64.zip" file from https://processing.org

Highlight the zip file and "Extract all" files to a suitable Destination directory (process4 say)
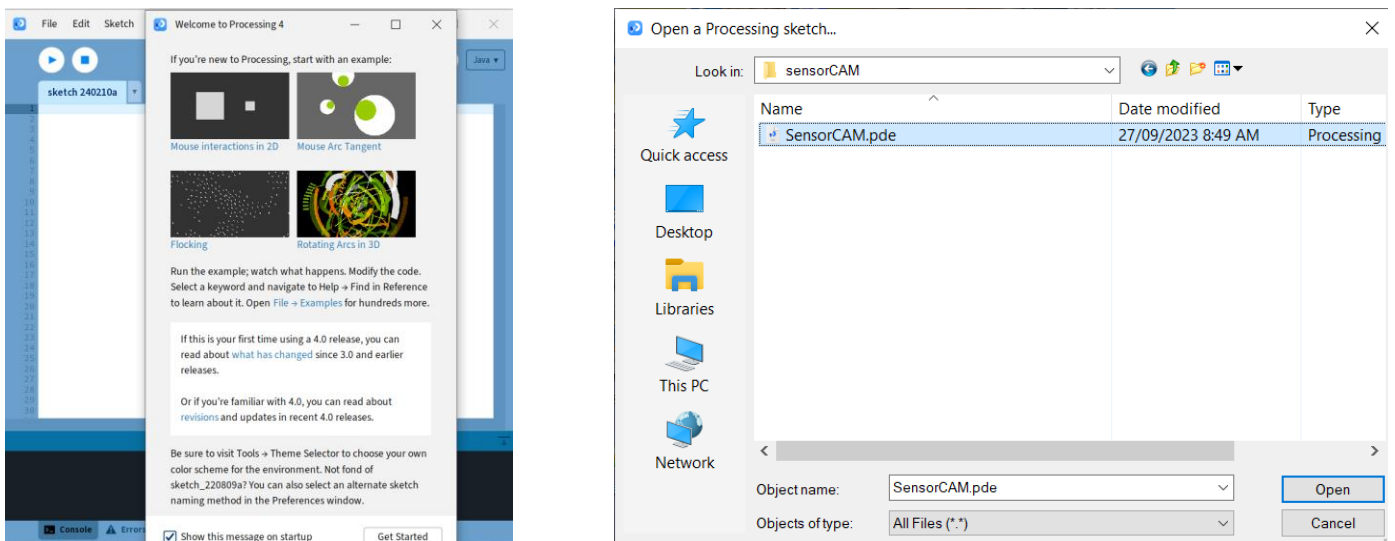
Locate and make a shortcut to the processing-4.3/processing.exe file on the desktop.

Make a sensorCAM folder in process4, Copy the sensorCAM.pde file into a process4/sensorCAM directory

Click the shortcut thereby opening Processing4

Click the GetStarted box on the Welcome window

Click file-open, select sensorCAM.pde and open.

Click the (left) circular run button to produce the test pattern shown below
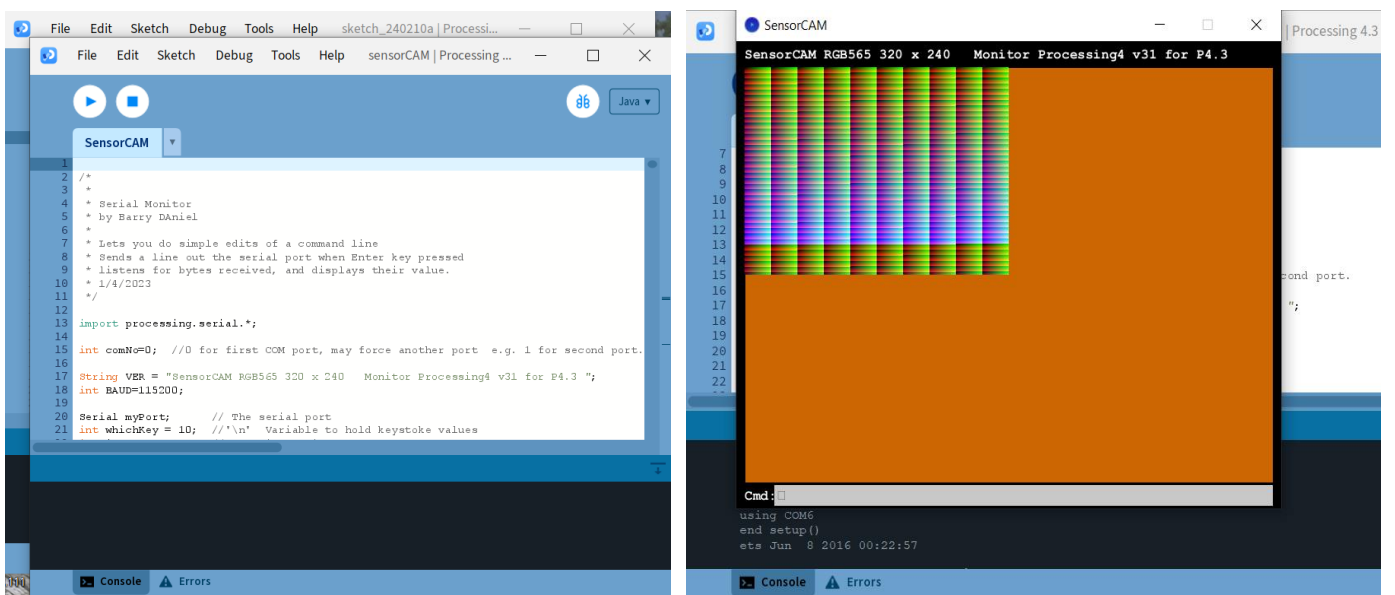
Reposition and resize the Console window and test pattern to look like that below, wide enough to avoid line wrap-around. The Cmd: line space will be the text entry line to be used for the CAM commands previously used with Arduino monitor. The broad black Console space replaces the sensorCAM monitor display and the test pattern window is for image display and sensor positioning.

Note the first lines of Console output. Choosing the first [0] COM port is the default. Should you have two (or more) COM ports listed, you may have to edit the *int comNo=0;* line to reflect the port the sensorCAM is attached to in the list (e.g. [1] as in image below). (this needs to be automated in a future revision)

If the sensorCAM reboots and runs, enter cmd: **w Enter** or **ver Enter** which will suspend any scrolling and print the sensorCAM version. Use the scroll bar to view the top/bottom of the Console window. Note that you may have to click the mouse cursor on the Cmd: window before typing commands.

Note the dark blue round icon on the taskbar will bring the image to the front should it vanish behind the Console window. The white round button to the right of "Run" will "Stop" the **sensorCAM.pde** program.
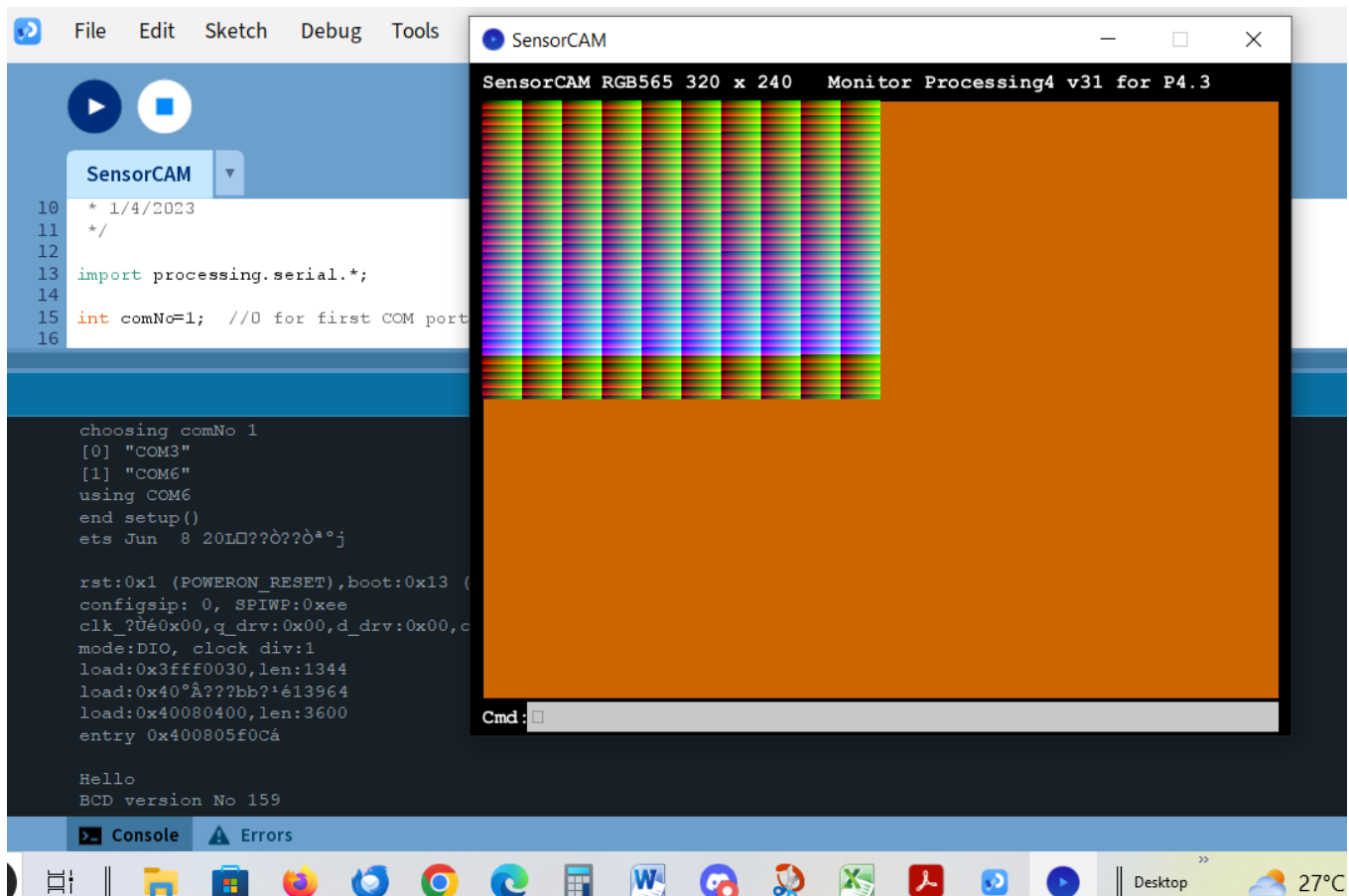
For convenience, locate sensorCAM/sensorCAM.pde and create a shortcut to this file on the desktop.

Optionally, the file Users/[Barry]/AppData/Roaming/Processing/preferences.txt can be edited with a text editor (e.g. notepad) for more convenient startup of sensorCAM.pde. Depending on your screen resolution, for convenience, adjust the window.height.default=700 and window.width.default=1200 as below. This should ensure that the opening Processing 4 window is higher and wide enough to avoid undesirable line-wrapping of scrolling sensor state tabulation. You can adjust these settings later if 700x1200 is not optimum for your screen resolution and sensor count.
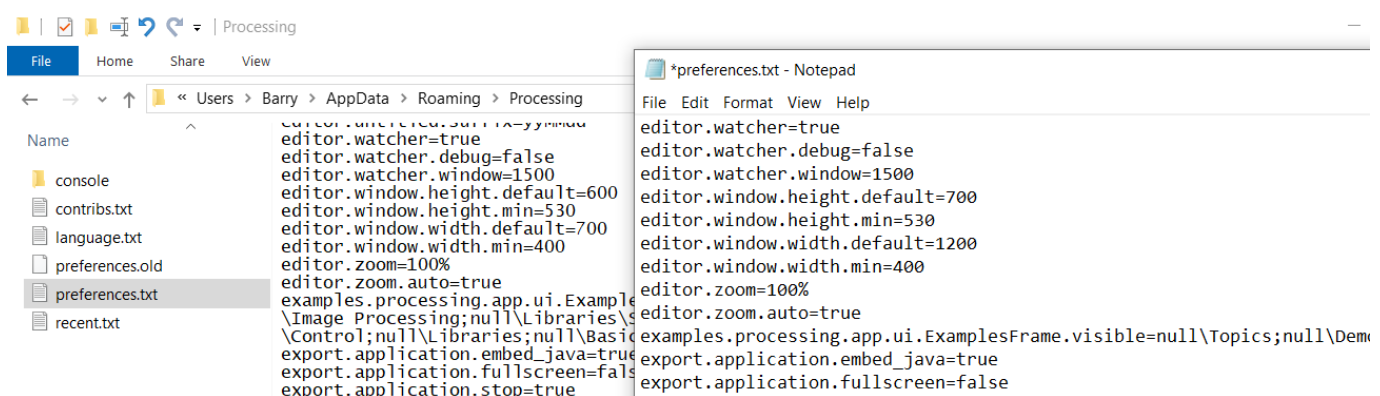
Step 7. Use the Processing 4 App to replace the Arduino monitor.

The processing app. can communicate the same commands to the sensorCAM as used in Step 5 above.

Proceed to verify the sensorCAM commands are generally working as before with Arduino IDE.

New commands can now be explored to see still images from the CAM and to see location of defined sensors as small squares. Enter the command 'Y' at the Cmd: line and wait 15 seconds for a full image to appear. Remember the sensors after rebooting will be those last stored in EPROM, not last defined.

To explore the imaging functions (W, X, Y, Z) in depth refer to the manual Chapter 6.
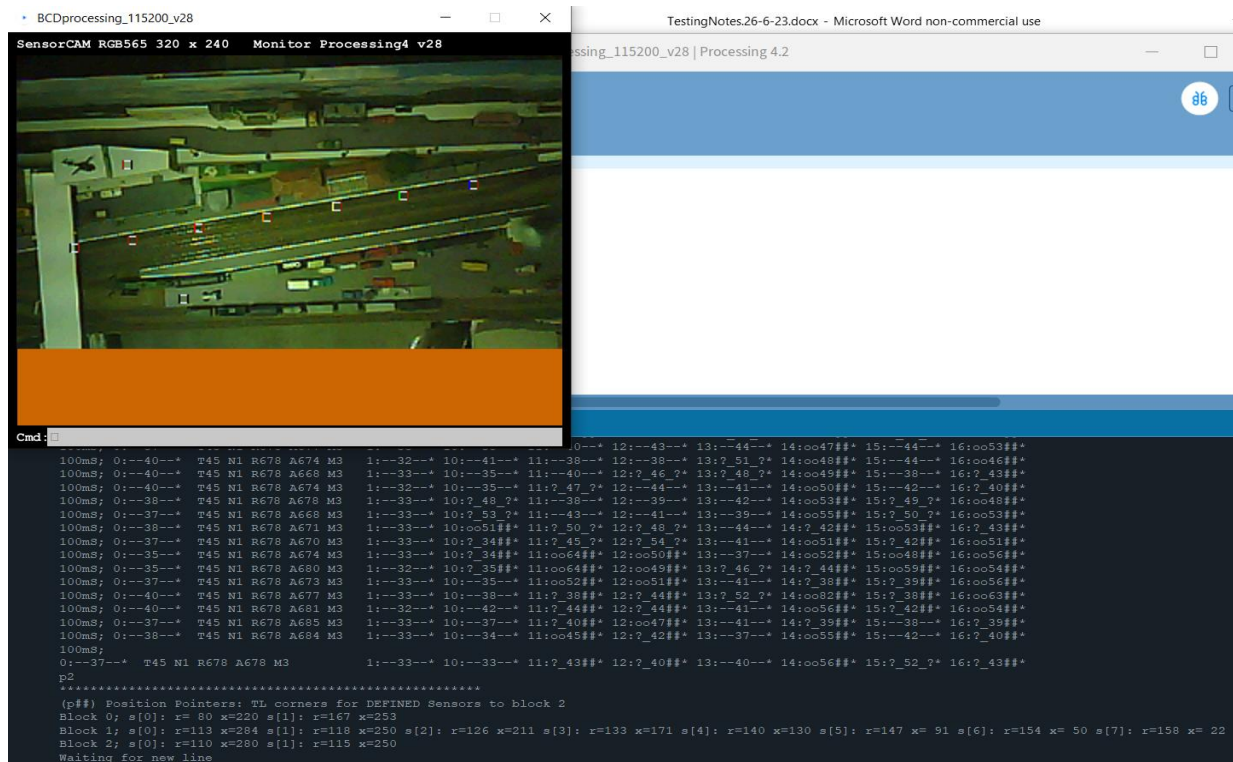
Step 8. Create test sensors and test detection with moving targets

Create some test sensors using Processing 4 by typing Cmd: **a31** (no Enter!) and following with a click on the image previously obtained as above. The command line will be completed with the cursor coordinates. Follow up with an **Enter** to define a new sensor S31. Verify the new sensor with a '**p3**' command and SEE the new sensor with a new '**Y**' command.

The sensor squares on the image are colour coded with the b/s number using the standard resistor colour codes (refer manual Appendix E).

Place a coloured object at the sensor position and observe the change in the scrolling tabulation for the sensor. Without the obstruction, after a reference command ('**r00**'), the difference score should be below 40 and with the obstruction the diff score should rise substantially. If the threshold is set ('**t45**' say) the sensor will "trip" and indicate occupied confirming successful installation. A moving hand test is an easy quick test to do.

It is advisable to define a "reference" sensor S00 that will NOT be obstructed. Any trips of this sensor will indicate an unwanted change to the environment (e.g. lighting). To define such a sensor use **a00** and a Processing 4 image "click" on a quiet location away from tracks but with "good" mid-range illumination (see Manual section 8.4). S00 is reserved and displayed separately at the left of the scrolling status table.

Step 9. Setup the CAM viewing a railroad and test a virtual sensor with moving rolling stock.

The CAM must be rigidly mounted, in a "convenient" location for testing at this stage. It may be limited by your cabling reach with USB control. Later, with buffered I2C and power, the CAMERA may be able to be located more optimally.

The manual Chapter 5 covers many points of relevance to getting satisfactory detection. Review this chapter and test topics for clarification. Good familiarity leads to success. Initial tests should be with good contrast between track and rolling stock. Refine parameters later to handle more difficult targets.

Step 10.   Optimise parameters for best performance.

Initially by trial & error you will learn the significance of parameters. This is an Alpha release and consequently the default settings may not be best. Refer manual Appendix B.

## Step 11.   Connect CAM to an i2c interface (e.g. PCA9515A or better)

The quickest and simplest method is to fit a PCA9515A to the ESP32-CAM-MB. Refer to the manual Chapter 3, the pictures in the manual (Figure 7 and Appendix F). Level shifting (3.3V to 5V) is the primary function of the PCA9515A. If distance becomes an issue, as the PCA9515A is limited to about 3metres, a buffered i2c solution will be needed. The manual Chapter 7 describes the prototype solution used. Other methods can be devised.

Step 12.   Depending on system, integrate sensorCAM into Command Station using appropriate code.
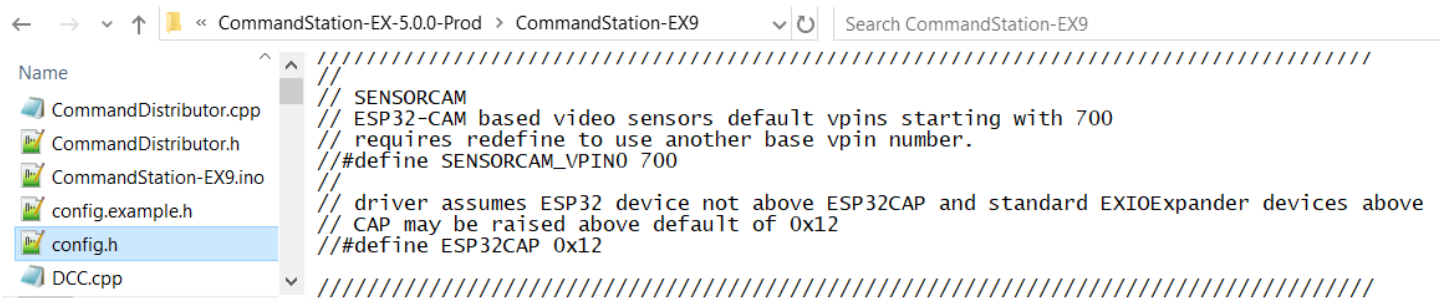
The CAM can be connected to a "Control Stations" via i2c using appropriate host code (typically C++). It has been successfully interfaced to two different (Arduino Mega) systems but most focus has been on the EX-Command Station (CS). The sensorCAM can be connected to a CS and interrogated using EX-Rail.

If you are wanting to integrate with a CS, we assume you have installed EX-CS previously. If not, refer to the DCC-EX website for details. The following guide assumes you have a working CS in place.

12.1 The sensorCAM repository has an EX-CS directory. These files normally reside with the *CommandStation-EX.ino* beside the other standard CS files. Copy *IO_EXSensorCAM.h* to this CS directory
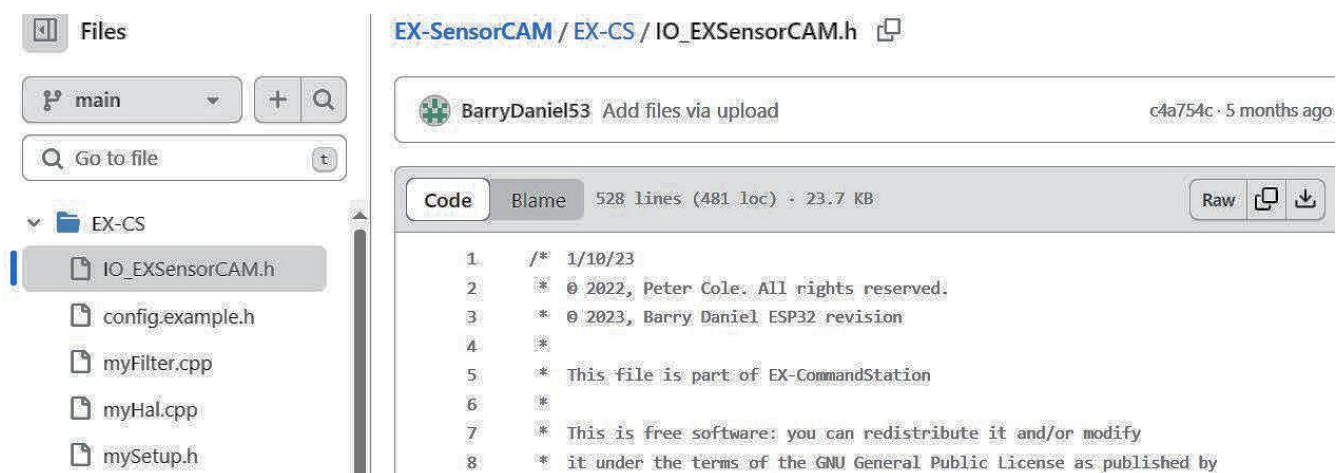
12.2 IF you do NOT want to use default CS addresses 700 to 780 for the sensorCAM virtual sensors, edit the config.h file to include *#define SENSORCAM_VPIN0 600* (say). Be careful to not change any other CS config lines from your previous working CS installation.
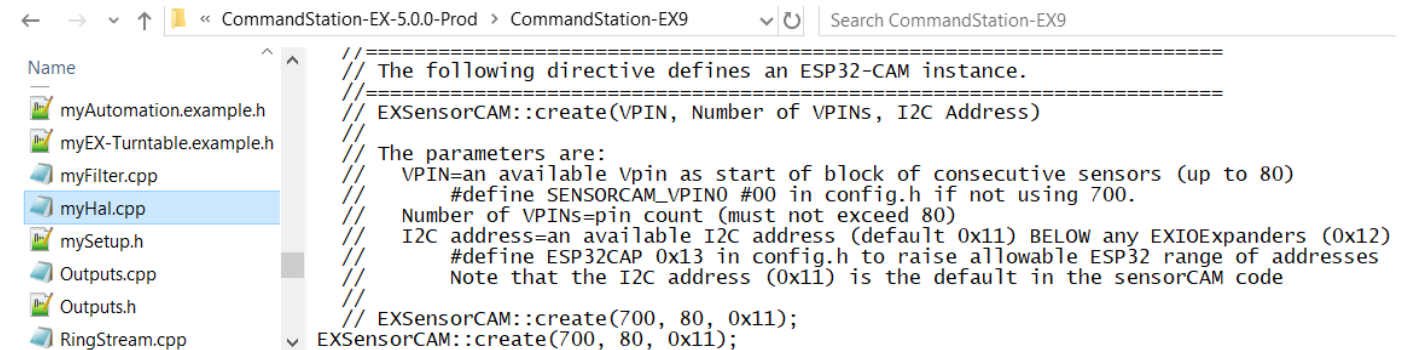
```
//////////////////////////////////////////////////////////////////////////////////////////////////
//
// SENSORCAM
// ESP32-CAM based video sensors default vpins starting with 700
// requires redefine to use another base vpin number.
//#define SENSORCAM_VPIN0 700
//
// driver assumes ESP32 device not above ESP32CAP and standard EXIOExpander devices above
// CAP may be raised above default of 0x12
//#define ESP32CAP 0x12
//
//////////////////////////////////////////////////////////////////////////////////////////////////
```

12.3 Edit myHal.cpp to tell CS to include/create a driver for sensorCAM. Add the sensorCAM driver with *#include "IO_EXSensorCAM.h* along with the other "includes" at the top. If not present, add the line *I2CManager.setClock(100000);* below *void halSetup() {* . Now add the "create" directive below. If you change *SENSORCAM_VPIN0* (step 12.2), then you need to amend the "create" from 700 to desired.
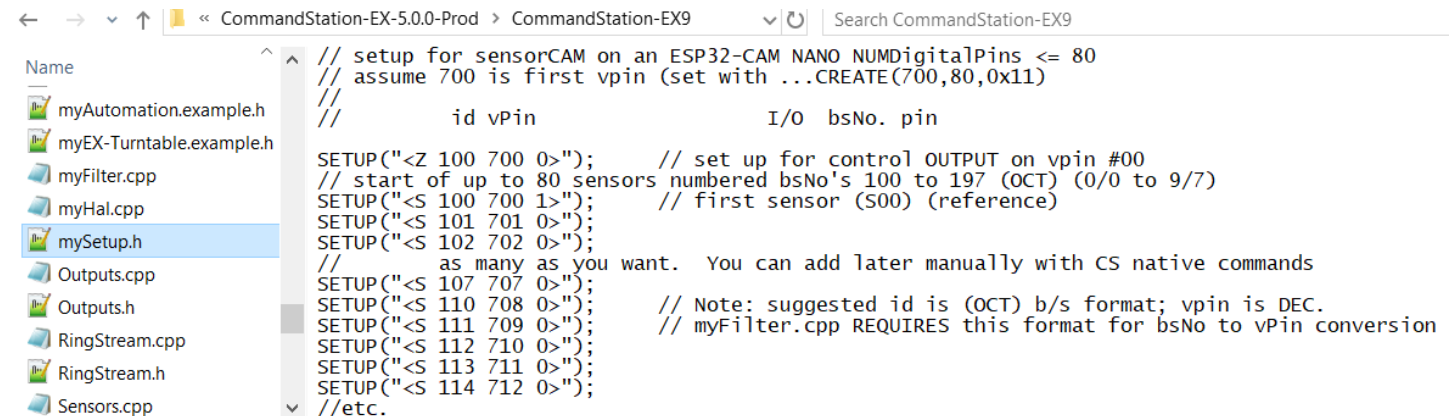
```
//===========================================================
// The following directive defines an ESP32-CAM instance.
//===========================================================
// EXSensorCAM::create(VPIN, Number of VPINs, I2C Address)
//
// The parameters are:
//    VPIN=an available Vpin as start of block of consecutive sensors (up to 80)
//        #define SENSORCAM_VPIN0 #00 in config.h if not using 700.
//    Number of VPINs=pin count (must not exceed 80)
//    I2C address=an available I2C address (default 0x11) BELOW any EXIOExpanders (0x12)
//        #define ESP32CAP 0x13 in config.h to raise allowable ESP32 range of addresses
//        Note that the I2C address (0x11) is the default in the sensorCAM code
//
// EXSensorCAM::create(700, 80, 0x11);
EXSensorCAM::create(700, 80, 0x11);
```

12.4 Now the individual sensor IDs need to be assigned to vPin numbers through the mySetup file. They can be added in manually, so we don't have to have all 80 defined now.
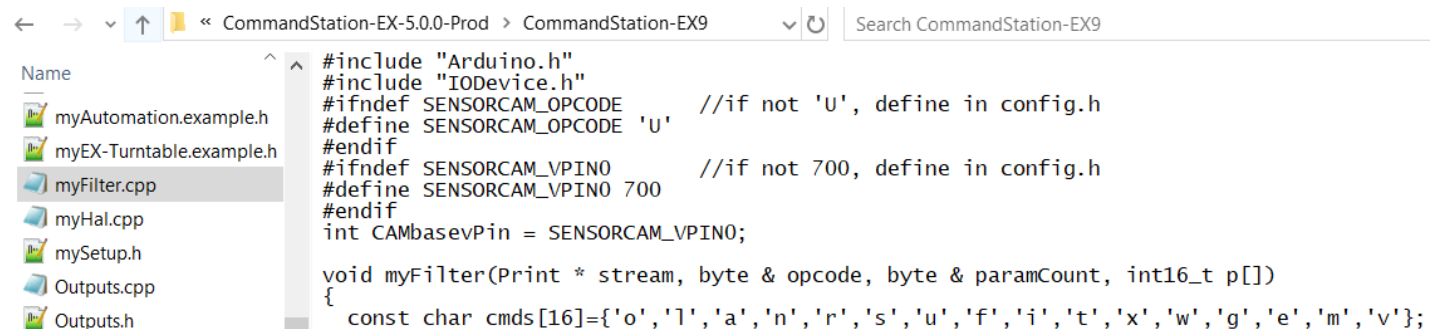
```
// setup for sensorCAM on an ESP32-CAM NANO NUMDigitalPins <= 80
// assume 700 is first vpin (set with ...CREATE(700,80,0x11)
//
//        id vPin                    I/O  bsNo. pin
//
SETUP("<Z 100 700 0>");      // set up for control OUTPUT on vpin #00
// start of up to 80 sensors numbered bsNo's 100 to 197 (OCT) (0/0 to 9/7)
SETUP("<S 100 700 1>");      // first sensor (S00) (reference)
SETUP("<S 101 701 0>");
SETUP("<S 102 702 0>");
//        as many as you want.  You can add later manually with CS native commands
SETUP("<S 107 707 0>");
SETUP("<S 110 708 0>");      // Note: suggested id is (OCT) b/s format; vpin is DEC.
SETUP("<S 111 709 0>");      // myFilter.cpp REQUIRES this format for bsNo to vPin conversion
SETUP("<S 112 710 0>");
SETUP("<S 113 711 0>");
SETUP("<S 114 712 0>");
//etc.
```

12.5 To provide CS with the ability to handle manual sensorCAM command entry (sent to sensorCAM via i2c instead of USB) a CS filter needs to be added. Assuming there is not already a myFilter.cpp file, copy the *myFilter.cpp* from Github to the CS directory. It is possible to combine two filters with care.

```
#include "Arduino.h"
#include "IODevice.h"
#ifndef SENSORCAM_OPCODE      //if not 'U', define in config.h
#define SENSORCAM_OPCODE 'U'
#endif
#ifndef SENSORCAM_VPIN0       //if not 700, define in config.h
#define SENSORCAM_VPIN0 700
#endif
int CAMbasevPin = SENSORCAM_VPIN0;

void myFilter(Print * stream, byte & opcode, byte & paramCount, int16_t p[])
{
    const char cmds[16]={'o','l','a','n','r','s','u','f','i','t','x','w','g','e','m','v'};
```

12.6 Once the changes have been put in place, the CommandStation.ino file will have to be recompiled and loaded into the CS (Mega?) using the Arduino IDE, to invoke the changes. Refer to the sensorCAM manual Appendix G for details on using the CS monitor to replace the sensorCAM USB monitor. Note that there are differences in the capabilities and format of display. No visual imagery can be obtained (except webcam) once the USB is disconnected. The CAM MUST be flashing to respond correctly to CS <U> cmds.

12.7 After up-loading to the Command Station, open the Arduino Monitor (at 115200 baud) to see the CS "console". If the sensorCAM i2c is connected and the CAM running (flashing), The user <U w> CS command should issue the CAM wait command (stops flashing) and another <U w> should resume. Note it can be helpful to have 2 instances of the IDE, so you can separately monitor CS and CAM on two USB ports.

```
<* License GPLv3 fsf.org (c) dcc-ex.com *>
<* I2<* License GPLv3 fsf.org (c) dcc-ex.com *>
<* I2C clock speed set to 100000 Hz *>
<* I2C Device found at 0x11, ?? *>
<* I2C Device found at 0x65, ?? *>
<* EX-IOExpander device found, I2C:0x65, Version v0.0.23 *>
<* EX-SensorCAM device found, I2C:0x11, Version v0.1.59 *>
<* MCP23017 I2C:0x20 Device not detected *>
<* MCP23017 I2C:0x21 Device not detected *>
<* Found PORTB pin 13 *>
<* Pin 55 Max 1497mA (501) *>
<* Found PORTB pin 12 *>
<* Pin 54 Max 1497mA (501) *>
<= A MAIN>
<= B PROG>
<iDCC-EX V-5.0.0 / MEGA / STANDARD_MOTOR_SHIELD G-3bddf4d>
<* SETUP("<Z 100 700 0>") *>
<O>
```

With CAM flashing, try <U i 11> and <U t 46> to confirm functionality and <U m> for current min/max settings. (Note <U t> command returns the LAST value of threshold. Repeat just to confirm change, if necessary, or use <U m>). Cmd <D HAL SHOW> will show the Vpin usage and version of sensorCAM.

```
<* CAM: w =251 *>
<* CAM: x =250 *>
<* CAM: i 11   *>
<* (i%%[,%%]) Info: Sensor 011(9) enabled:1 status:0   row=118 x=250 Twin=00 A~1728 *>
<* CAM: t 46   *>
(t[##[,##]]) Threshold:40 sensor S00:-32--*
01:--32--* 10:--33--* 11:--33--* 12:--33--* 13:--33--* 14:--34--* 15:--33--* 16:--32--*
<* CAM: t 46   *>
(t[##[,##]]) Threshold:46 sensor S00:-32--*
01:--33--* 10:--34--* 11:--33--* 12:--33--* 13:--33--* 14:--32--* 15:--33--* 16:--33--*
```

```
        <* CAM: m =254 *>
        <* (m$[,##]) Min/max: $ frames min2flip (trip) 3, minSensors 00, maxSensors 024, nLED 1, threshold 46
<U m>      ...
```

```
                <* Arduino Vpins:2-69 *>
                <* EX-IOExpander I2C:0x65 v0.0.23 Vpins 800-817   *>
                <* EX-SensorCAM I2C:0x11 v0.1.59 Vpins 700-779   *>
                <* PCA9685 I2C:0x40 Configured on Vpins:100-115 OFFLINE *>
<D HAL SHOW>    <* PCA9685 I2C:0x41 Configured on Vpins:116-131 OFFLINE *>
```

12.8 To use EX-RAIL refer to the EX documentation.