# Optical Sensor Calibration and Testing

This document assumes that you have installed at least the Matlab software as described in [Arduino-Matlab Interface Software.docx](Arduino-Matlab%20Interface%20Software.docx). In particular the arduinoPort property must be set appropriately in RigParameters.m; all other properties are not used. None of this code depends on [ViRMEn](http://virmen.princeton.edu/).

## Image Quality

The image quality from the optical sensor should be maximized in order for it to obtain reliable measurements, which can be achieved by:

* Ensuring sufficient infrared (IR) illumination (but increasing it past a certain threshold does not help).
* Adjusting the focal plane of the M12 lens to be at the surface to be measured.
* Using a surface with more texture (of the appropriate size given the limited number of sensor pixels).

The ADNS-3080 chip reports a SQUAL value that can be read out using software as explained below. This value needs to be at least 30 if the velocities to be measured are around 100cm/s-150cm/s. For lower values of SQUAL, the sensor tends to under-measure the actual displacement, with the size of the effect increasing with higher surface velocities. The following are examples of images of a Styrofoam ball at various SQUAL values ranging from unusable to ideal (for velocities not far exceeding 100cm/s):

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| Unusable | **Marginal** | **Good** | **Ideal** |

Procedure:

1. Load Arduino Code\ADNS\_image\_v1\ADNS\_image\_v1.ino in the Arduino IDE.
2. Edit the reset\_pin and select\_pin values (lines 20-24) to select the sensor of interest:

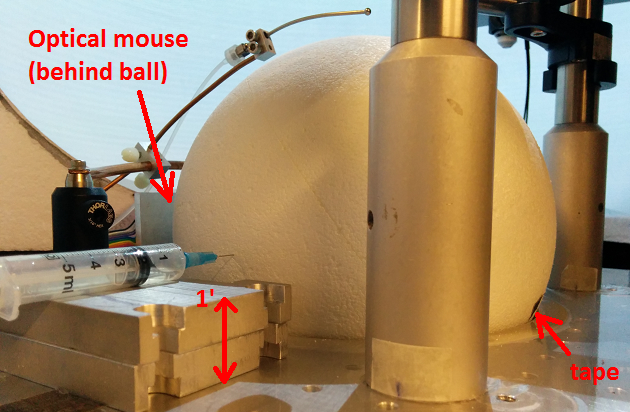
|  |  |  |
| --- | --- | --- |
|  | Sensor 1 | Sensor 2 |
| reset\_pin | 8 | 6 |
| select\_pin | 4 | 10 |

1. Upload the Arduino code to the board. Note that you will have to reprogram the board after this to use it as a displacement readout.
2. Print the left side of the following image (an optical spoke target) using a laser printer. The spoke target consists of lines of vanishing size towards the center, therefore allowing you to probe the single pixel limit of the optical sensor as shown in the calibrated image (from the sensor) in the right:

|  |  |
| --- | --- |
| C:\Users\GS70\AppData\Local\Microsoft\Windows\INetCache\Content.Word\spokes.eps |  |

1. Find some way of placing the spoke target in front of the optical sensor at the same location and orientation as the actual surface to be measured. For example for the mouse virtual reality rig it can be taped onto a Styrofoam ball and the ball can be suspended at the height it would usually be at during experiments.
2. Run the Matlab Code\Calibration\display\_image.m function. It is programmed to continuously display the sensor image for a predetermined amount of time, and can be terminated by pressing <Ctrl+C>.
3. Adjust the IR LED so that it points towards the center of the surface to be imaged. You should see the illuminated region shift around in the display\_image figure.
   * The display\_image figure is normalized so that the brightest regions appear white and the darkest regions appear black, i.e. absolute luminosity information is not available. You can change this behavior by editing the code to replace imagesc(im) (line 66) with image(im) and setting the color scale manually: set(gca, 'CLim', […])
   * Oblique illumination can help increase feature contrast on uneven surfaces, but in general just illuminating the largest amount of surface available is sufficient.
4. Adjust the focus of the optical sensor lens by rotating it. You should see the spoke target lines get sharper and the SQUAL value increase if you’re going in the right direction. You should be able to find a distance at which the SQUAL value is maximal (there is a decent amount of leeway).
5. Replace the spoke target with the actual surface to be measured, and verify that the SQUAL value is still high enough (>> 30). Values of 50-80 have been achieved with Styrofoam balls, the higher end if they have been “weathered” by running mice.
6. If the SQUAL value is 30-ish or worse, consider using a more textured surface. For example, scour the Styrofoam ball with steel wool.

## Length Scale

1. Load Arduino Code\ADNS\_aout\_wUSB\_1sensor\ADNS\_aout\_wUSB\_1sensor.ino in the Arduino IDE.
2. Edit the default\_sensor\_pin value (line 12) to select the sensor of interest (e.g. 4 for the first sensor, 10 for the second sensor, if any).
3. Upload the Arduino code to the board.
4. Mouse virtual reality rig: Suspend a Styrofoam ball on an axle, or a cylindrical Styrofoam wheel, at the position it would be in relative to the optical sensor in a real experiment.
   * For other rigs, you will have to come up with a way to generate a known amount of displacement of the surface to be measured.
5. Mark a reference position on the ball, e.g. with a piece of black tape as shown in the mouse VR calibration photo to the right.
6. Ensure that Matlab Code\Calibration\calibrateBall.m is either in the current folder in the Matlab console, or otherwise in the Matlab path, before continuing the rest of this procedure.
7. While holding the ball in place, run the calibrateBall script.
8. Spin the ball 10 times (this can be done quickly), counting the number of times that the black tape returns to approximately its original location.
   * If necessary, a small correction can be made to rotate the ball further until the tape is exactly at its original position. This should cancel out errors incurred in the previous step.
9. Stop calibrateBall by pressing Ctrl+C.
10. Input fclose(instrfindall) at the Matlab command line to close Arduino communications.
11. Input [dx,dy] at the Matlab command line to view the accumulated displacements. If the optical sensor has been correctly aligned w.r.t. the axis of rotation of the ball, one of these displacements should be large (> thousands) and the other one small (< tens).

In the following the measured displacement along the axis of interest will be referred to as nDots, and the number of revolutions used as nRev.

1. Repeat this measurement until satisfied (with more rotations if necessary), then enter the obtained constants into RigParameters.m:

|  |  |
| --- | --- |
| RigParameters | Description |
| .ballCircumference | Actual displacement (e.g. in cm) of the calibration surface per revolution. For an 8-inch diameter Styrofoam ball, this is its circumference, 63.8cm. |
| .sensorDotsPerRev | This should be set directly to nDots/nRev in the case of a single sensor. In the case of two sensors, use the code  sensorDotsPerRev = RigParameters.sensorCalibration()  and set the latter as described below. |
| .sensorCalibration() | This is only used for two sensors, which may have differing constants. In this case use the appropriate MovementSensor label for the sensor of interest to record dotsPerRev, e.g.:  dotsPerRev(MovementSensor.FrontVelocity) = nDots/nRev; |

## Analog Output Scaling

The Arduino programs ADNS\_aout\_wUSB\_1sensor.ino and ADNS\_aout\_wUSB\_2sensors.ino causes the microcontroller to continually update the voltage of two analog output ports (BNC terminals on the enclosure) as follows:

|  |  |
| --- | --- |
| Program | Analog output |
| ADNS\_aout\_wUSB\_1sensor.ino | Accumulated x and y displacement values measured by the optical sensor in the last polling period. |
| ADNS\_aout\_wUSB\_2sensors.ino | Like above, but for the second sensor by default. This behavior can be changed by commenting in/out lines 171-172 and/or 203-204, which look like:  analogWrite(DAC0, map(x\_motion,-127,127,0,4095));  analogWrite(DAC1, map(y\_motion,-127,127,0,4095));  The two analog channels are DAC0 and DAC1 respectively, and the values that they can be updated with include x\_motion/y\_motion which correspond to displacements for sensor #1, and x\_motion\_2/y\_motion\_2 which correspond to displacements for sensor #2. |

For example when recording the above analog signals, it can be desirable to scale the voltage signal to convert it to physical units, e.g. the surface movement velocity. To do this one needs to know:

1. The analog output resolution.

By default this is set to 12 bits using the command analogWriteResolution(12) in the setup() function of the Arduino programs. This means that all further analogWrite() commands in the code use the map() function to convert the quantity to be output to a range between 0 and 212-1 = 4095. Setting a value of 0 (4095) via analogWrite() will cause the analog output to switch to the lowest (highest) possible output voltage. If you want to change the output resolution, make sure to consistently change all the ranges of the analogWrite() functions in the code.

1. The minimum/maximum output voltage of the Arduino board.

This range is measured to be [0.55V, 2.76V] for an Arduino Due board. To measure it for your Arduino, and if you have a data acquisition device that can record a waveform:

1. Uncomment the last 9 lines in the setup() function in either ADNS\_aout\_wUSB\_\*.ino programs, i.e. the ones starting with analogWrite(DAC0, 0).
2. Connect at least one analog output port of the Arduino to your data acquisition card and start acquiring its waveform.
3. Upload the Arduino program. Immediately after the program has been uploaded, the setup() function will execute.
4. You should see a fast reset blip in the analog output waveform followed by three rectangular steps lasting 2 seconds each, which correspond to the minimum (V-), middle (V0), and maximum (V+) output voltages that the Arduino board is capable of.
5. Stop data acquisition and record the values of V0 and V+. You can use V- to verify that the scaling is linear, as it should be: V0 = (V- + V+)/2.
6. Remember to comment out the 9 lines in the setup() function and re-upload the original code to the Arduino. Otherwise the delay and output from setup() can cause undesirable effects at the start of your experiment.

If you only have a non-recording oscilloscope (with two channels), you can still obtain this information by using only the following analogWrite() commands in the setup() function:

analogWrite(DAC0, 0);

analogWrite(DAC1, 4095);

and removing the code in the loop() function. This will cause the Arduino to output a constant voltage so as to be easy to read.

1. The physical distance per sensor readout unit (dots), as measured in the length scale calibration section above. For example if a value of nDots was recorded by the sensor by spinning a ball nRev times, and the ball has circumference L, then the distance per dot is

distancePerDot = L / (nDots/nRev)

For angular measurements, one can use L = 2π.

1. The sensor polling interval, poll\_interval as set in the Arduino code. This is 10ms by default. When the sensor readout is to be interpreted as a velocity, the scaling is:

vPerDot = distancePerDot/poll\_interval = L/(nDots/nRev)/poll\_interval

1. The readout range of the ADNS-3080 sensor, which is a signed 8-bit integer, i.e. -127 for the largest possible “backwards” displacement and +127 for the largest possible “forward” displacement.

The following calculation works out the appropriate analog voltage conversion factors for the Clampex data acquisition software:

* Using the menu item Configure 🡪 Lab Bench, a scale factor and offset can be set per channel, such that Clampex will apply a linear transformation to its recorded (input) voltage and display:

channelValue = offset + inputVoltage / scaleFactor

* The sensor readout is mapped linearly onto the Arduino output voltage range:

|  |  |  |  |
| --- | --- | --- | --- |
| Number of dots measured by sensor | -127 | 0 | +127 |
| Corresponding physical velocity | -127 / vPerDot | 0 | +127 / vPerDot |
| Analog output voltage | V- | V0 | V+ |

* In order for the displayed channelValue to be the physical velocity, we have the following two equations:

|  |  |  |
| --- | --- | --- |
| 0 | = | offset + V0 / scaleFactor |
| +127/vPerDot | = | offset + V+ / scaleFactor |

* Solving the above yields:

|  |  |  |
| --- | --- | --- |
| scaleFactor | = | (V+ - V0) / (127/vPerDot) |
| offset | = | -V0 / scaleFactor |

## Readout/Transmission Speed

The optical sensor readout code (ADNS\_aout\_wUSB\_\*.ino) polls the sensor at intervals specified by the poll\_interval variable (around line 22). This is by default 10ms. The actual readout rate for a two-sensor setup can be tested using the following program:

Arduino Code\ADNS\_aout\_wUSB\_2sensors\_speedtest\ADNS\_aout\_wUSB\_2sensors\_speedtest.ino

The above code is the same as ADNS\_aout\_wUSB\_2sensors.ino (so that the processing load is as similar as possible) except for the addition of some lines that will be explained below. An analogous modification can be made to ADNS\_aout\_wUSB\_1sensor.ino to test the single-sensor setup.

### Test #1

1. Line 23 defines the variable testBit, which is toggled between 0 and 1 at every poll\_interval.
2. Uncomment lines 218-219 and comment out lines 222-223 in order to update the analog output values with testBit at every poll\_interval. The uncommented lines should look like:

// Test #1

analogWrite(DAC0, map(testBit,0,1,0,4095));

analogWrite(DAC1, map(testBit,0,1,0,4095));

1. Connect the analog output lines of the Arduino optical sensor box to an oscilloscope or equivalent data acquisition device.
2. Upload ADNS\_aout\_wUSB\_2sensors\_speedtest.ino to the Arduino board.
3. You should see (two) square waves on the oscilloscope, with period 2×poll\_interval.

### Test #2

1. Comment out lines 218-219 and uncomment lines 222-223 in order to update the analog output values with the optical sensor readouts at every poll\_interval. The code is currently set up to transmit one value from each sensor, i.e. displacement in the y direction, but you can change this to displacements in x as well. The uncommented lines should look like:

// Test #2

analogWrite(DAC0, map(y\_motion ,-10,10,0,4095));

analogWrite(DAC1, map(y\_motion2,-10,10,0,4095));

1. Connect the analog output lines of the Arduino optical sensor box to an oscilloscope or equivalent data acquisition device.
2. Put something for the sensors to measure that has naturally occurring small movements. For the mouse VR application, a Styrofoam ball floating on an air cushion works well as it is unstable and will wobble around without external force.
3. Upload ADNS\_aout\_wUSB\_2sensors\_speedtest.ino to the Arduino board.
4. You should see occasional changes in the waveform on the oscilloscope, with temporal discreteness at multiples of poll\_interval. For very small motions like the ball on an air cushion, this should look like small square pulses (corresponding to the smallest measurable unit of displacement) with duration poll\_interval, or rarely 2×poll\_interval, etc.