This folder contains information for assembling an optics-based velocity sensor for use on virtual reality (VR) systems. The system is based on an ADNS 3080 optical mouse sensor used in conjunction with an Arduino Due microcontroller. Illumination is provided with an infrared LED at 890nm. The system is designed to replace the use of an “optical mouse” as a velocity sensor in VR systems. The principal advantages over the optical mouse are:

1. Lenses can be used that provide greater depth-of-focus for the image focused onto the sensor chip; thus small changes in the distance of the sensor and the treadmill used in VR (commonly supported on a cushion of air) produced by mouse running, etc., do not cause problems with the sensing of motion, through blurring of the image.
2. An image of the surface taken through the lens and sensor can be directly observed using a Matlab program, which helps in optimally setting up the optical system, and a numerical performance metric termed SQUAL for “surface quality” can be monitored in real time. SQUAL is proportional (1/4 X) to the number of visual features on the surface that is imaged that are used in the correlation based tracking algorithm implemented on the chip.
3. Sensor readout rates can be controlled by the Arduino programmer and velocities easily read by the VR computer without the use of any other electronics or software. Currently the implementation we use is based on a 100Hz readout, i.e. treadmill velocity is sampled every 10ms, which is used to update a running total of motion locally stored on the Arduino. The running totals for X and Y velocities for up to two ADNS 3080 sensors can be read asynchronously by a host computer implementing serial communication with the Arduino Due through a USB interface.

The ADNS 3080 sensor that we currently use is mounted on a small circuit card that was constructed for use in drones as a sensor for monitoring velocity over the ground. It goes by the name “APM 2.52/2.6”. An “S-mount” lens (M12 x 0.5 thread; also known as an “M12 lens”) focuses light onto the sensor. The card typically comes with a f=4.2mm focal length lens; we use that standard lens for mounting the sensor near the exposed equator of a spherical treadmill (the mouse is at the “north” pole) with the lens “aimed” directly at the surface. For mounting the sensor under the treadmill (at the “south” pole of the sphere), we cut a hole in the bottom of our spherical treadmill base cup and cover the hole it with a circular glass window. The sensor is then mounted with the optical axis horizontal (90 degrees to the north-south axis of the sphere) and it images the bottom of the sphere through a 90-degree mirror located directly below the glass window. We find that velocity sensing with this 90-degree optical axis geometry is better performed using a different M12 lens for the 3080 chip; we use a f=8mm focal length lens from Edmunds Optics. M12 mount lenses are focused by rotating the lens in the lens mount. The APM 2.52 card with the ADNS 3080 does not come with an illumination source. So we use an Optek OP290A IR LED for illumination; it is mounted on some standoffs mechanically attached to the APM circuit card and the LED runs off of the 3080 chip power supply line. The leads of the LED are left long and they can be bent to direct the LED light at the target to help maximize the SQUAL metric described earlier.

Communication with the host PC running VR software is very simple conceptually during normal operation. The host computer sends the character “m” to the Arduino over the USB serial port whenever it wants movement information, for example at the beginning of the time period for calculating the next VR display frame. The Arduino responds by returning delta-X ; delta-Y , followed by a CR (carriage return) on the serial USB line to the PC. Here delta-X is the sum of all X movement values the Arduino has received every 10 ms from the ADNS 3080 since the last movement request from the VR computer; a similar running total is communicated by delta-Y for Y motion. The running totals on the Arduino are zeroed after communication to the PC and the accumulation starts again on the Arduino. In addition to the accumulation of movement information on the Arduino, the code on the Arduino also directs the value of each new X and Y motion reading from the ADNS 3080 to a pair of Digital to Analog (D/A) converters. Thus there are analog voltage signals that change every 10 ms and that represent the movement information on the X and Y channel of the optic sensor. You can simply hook up an oscilloscope or other recorder and monitor the movement signals. The scaling of the analog voltages is a little unconventional since the Aruduino Due runs on a 5V unipolar power supply, and is explained in our documentation.

In this folder you will find:

1.) A Folder with Arduino Code.

2.) A Folder with Matlab Code

3.) A Folder with Drawings and Datasheets

4.) A Folder with Instructions for Assembling Sensor/LED/Arduino and Installing and Running Software

Detailed contents of this package:

|  |  |
| --- | --- |
| README.docx | This file |
| Drawings and Datasheets |  |
| ADNS-3080\_datasheet.pdf | Optical flow sensor datasheet |
| OP290series.pdf | Infrared LED datasheet |
| Due\_ADNS\_connections\_single\_sensor.pptx | Arduino Due wiring diagram for single sensor |
| Due\_ADNS\_connections\_Dual\_Sensor.pptx | Arduino Due wiring diagram for two sensors |
| Part\_bottom.pdf | CAD drawing for bottom half of Arduino enclosure |
| Part\_top.pdf | CAD drawing for top half of Arduino enclosure |
| Lower\_Sensor\_Assembly.pdf | CAD drawing for assembly of plate holding lower sensor |
| Instructions |  |
| Parts List.xlsx | Part name, numbers and suppliers for all components |
| Arduino-based Optical Sensor Assembly Instructions.docx | Instructions for building the enclosure and wiring an Arduino board to one or two optical sensors |
| Arduino-Matlab Interface Software.docx | Instructions for software that programs the Arduino to transmit sensor readouts, and Matlab software to acquire this data |
| Sensor Calibration Method and Software.docx | Instructions for how to calibrate, scale, and test the optical sensor readout |
| Arduino Code |  |
| ADNS\_aout\_wUSB\_1sensor\ADNS\_aout\_wUSB\_1sensor.ino | Arduino program for transmitting data from one sensor |
| ADNS\_aout\_wUSB\_2sensors\ADNS\_aout\_wUSB\_2sensors.ino | Arduino program for transmitting data from two sensors |
| ADNS\_aout\_wUSB\_2sensors\_speedtest\ADNS\_aout\_wUSB\_2sensors\_speedtest.ino | Arduino program for testing the sensor readout rate |
| ADNS\_image\_v1\ADNS\_image\_v1.ino | Arduino program for transmitting the optical sensor image and SQUAL value (to Matlab Code\Calibration\display\_image.m) |
| Matlab Code |  |
| MovementSensor.m | Enumeration used to identify various sensors when the Arduino is connected to multiple sensors |
| RigParameters.m | Class with collection of rig-specific constants like the Arduino COM port number; this keeps the rest of the code the same for all rigs |
| example\_ViRMEn\_experiment.m | Example ViRMEn experiment code corresponding to instructions in [Arduino-Matlab Interface Software.docx](Instructions/Arduino-Matlab%20Interface%20Software.docx) |
| Calibration |  |
| calibrateBall.m | Matlab code to read out the raw sensor displacement values when calibrating it to a known physical distance travelled |
| display\_image.m | Matlab code to display the image and SQUAL value transmitted by the Arduino when calibrating readout quality |
| OneSensor |  |
| MouseReader\_1sensor.m | Class that handles communications with an Arduino transmitting information from one sensor |
| initializeArduino\_1sensor.m | Function to create a **MouseReader\_1sensor** object and set up required calibration constants |
| TwoSensors |  |
| MouseReader\_2sensors.m | Class that handles communications with an Arduino transmitting information from two sensors |
| initializeArduino\_2sensors.m | Function to create a **MouseReader\_2sensors** object and set up required calibration constants |
| movements |  |
| moveArduino.m | ViRMEn movement function for a one-sensor setup where the sensor is at the equator, i.e. reads out changes in pitch and yaw |
| moveArduinoLinearVelocity.m | ViRMEn movement function for a two-sensor setup where the sensor is at the south pole, i.e. reads out changes in pitch and roll; displacement is interpreted as velocity in the virtual world |
| moveArduinoLiteral.m | ViRMEn movement function for a two-sensor setup where the sensor is at the south pole, i.e. reads out changes in pitch and roll; displacement is interpreted as if the animal is in the lab frame |