**AirSAS Documentation**

**Julia Ng**

**Temperature Control Documentation**

**Block Diagram**

Computer

5

4

3

2

1

MATLAB

Raspberry Pi

PuTTY

LabVIEW

Descriptions:

Via a python script “temperature\_sensor.py”, the temperature probe (DS18B20) measures the ambient temperature of the room and returns a string containing the temperature in Celsius. The temperature probe and the Pi communicate via **One-Wire protocol**, and the CRC bit tells us if the data is being received correctly.

1

The LEDs will be used as indicators of the status of the system, i.e. if the temperature probe is not connected properly, if there is an error, et cetera. The Green LED will indicate if the Raspberry Pi and LabVIEW are communicating properly, and the Red and Orange LED will indicate any errors. Exactly how these errors will be indicated will be discussed later. The LEDs are connected by GPIO.

2

The Pi and the computer will be connected directly via Ethernet. In order to have communication between the Pi and LabVIEW, we will be using **TCP/IP** protocol. A python script will be running on the Pi on startup that is listening for connections on port 3000 at IP address 192.168.0.10, acting as a server. The LabVIEW VI, on the other hand, will create a connection with the Pi over these addresses whenever it is run, acting as the client. Once the VI is run, it will receive the temperature and date and time at the time it was requested in a string message that is formatted below. A more in-depth description of the system is also detailed below.

3

The Raspberry Pi will not always have a monitor attached to it; In fact, once integrated with the system, it will likely not have a monitor at all. Thus PuTTY (on the computer) can be used to **SSH** into the Pi once the two are connected on the same network. This will only be necessary if we need to debug something on the Pi itself, which will likely not be needed.

4

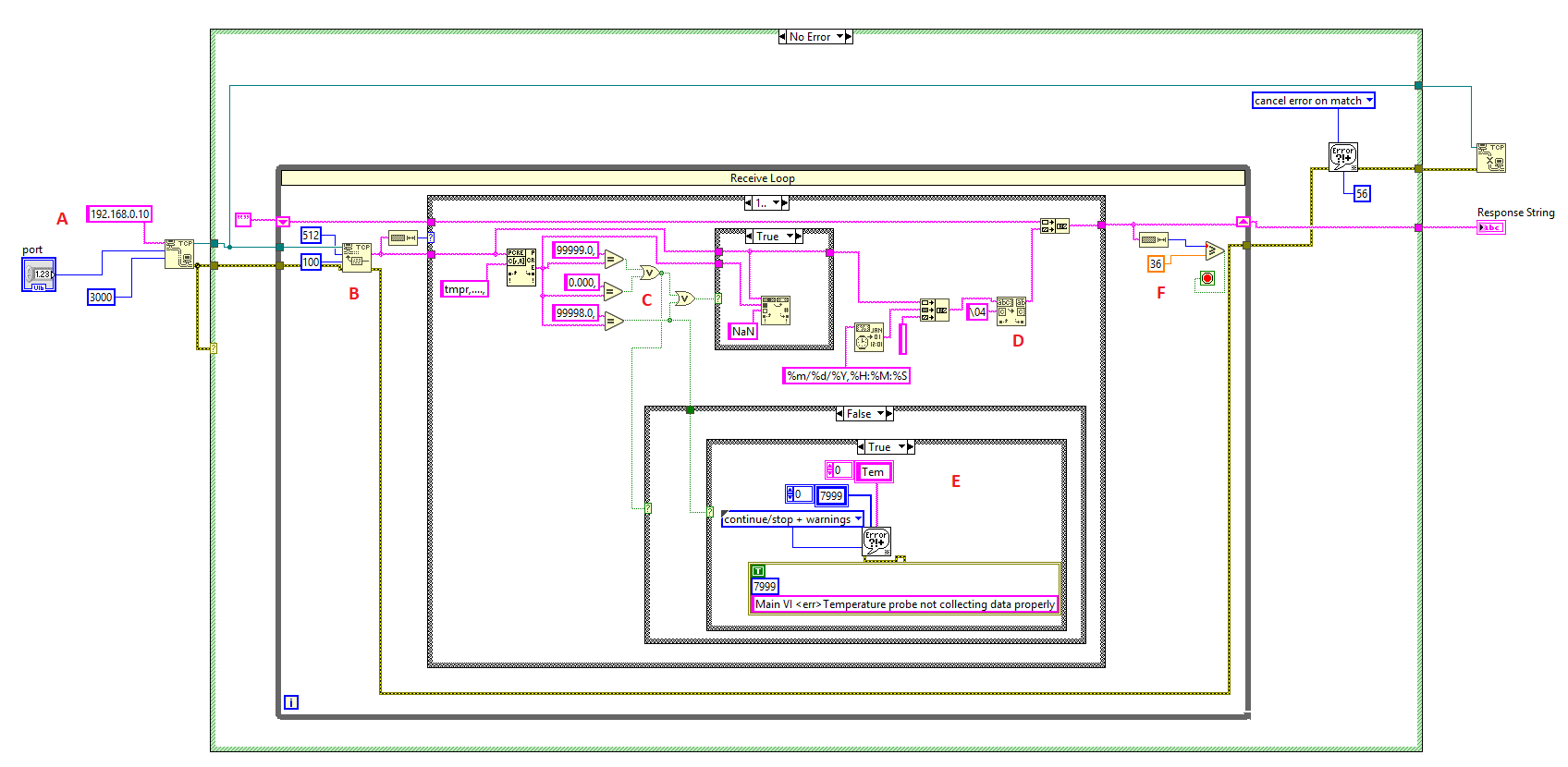
The message received can then be parsed, and the relevant data can be put into HDF5 files and used in MATLAB to beam-form.

5

**System Description**

LabVIEW Client

*Block Diagram*



1. Opening a connection between IP address 192.168.0.10 and Port 2056. This IP corresponds to the IP of the Pi, and this port number was picked randomly. 3000 ms (3 seconds) is the timeout for the server connection. If there are any errors at this stage, the connection is closed.
2. Reads the message that the Pi sends via TCP/IP. 512 specifies the byte size of the message, however this will rarely be exceeded. The message at this stage only contains the identifier, the raspberry pi identifier, and the temperature in Celsius. An example message that is being read is “tmpr,RPI1,25.000,”
3. This is handling the temperature probe specific errors. If the temperature message contains the string “99999.0,”,“99998.0,”, or “0.000,”, there is an error and “NaN” should be recorded as the temperature value. Any other number is considered valid.
4. This is formatting the message that is received. This concatenates the message like the one mentioned in (B) and adds the date, time, and then a return & newline character. This gives you a message in the format that was mentioned earlier.
5. This is the custom error handler for error 7999 (temperature = 99998.0). This error displays a custom error message – “Temperature probe not collecting data properly” along with the option to stop the VI or continue. A similar error handler exists for error 8000.
6. Closing the connection once the temperature string is received

Note: To find the TCP/IP blocks, go to View > Functions Palette > Data Connectivity > Protocols > TCP

*Error Codes*

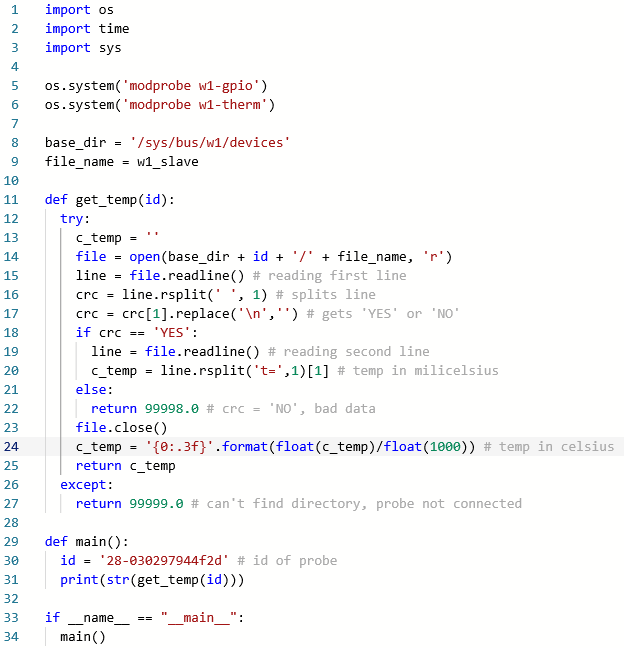
|  |  |  |
| --- | --- | --- |
| Error Code # | Message/Reason | Possible Fixes |
| 56 | Connection with server timed out | Stop the VI, reboot Pi, then re-run the VI client |
| 59, 63 | TCP open connection | Double check that the Pi and computer are connected by Ethernet, re-plug in  Check that the port number on VI matches code |
| 7999 | Temperature probe is not collecting data properly | Screw GPIO4 Pin in tighter  Rewire GPIO4 (yellow wire and resistor)  Rewire VDD (red wire and resistor)  Consult the hardware section if issues ensue (pg. 7) |
| 8000 | Temperature probe is not connected | Same as above (error 7999)  If error continues after trying all of the above, switch probe and modify code accordingly |

Python Server

*Design*

The computer initiates a request for the most recent temperature measurement, causing the Pi to run the temperature script and return the data to the terminal window.

*Temperature Sensor Code temperature\_sensor.py*

* Import the necessary modules
* Follow one-wire protocol and enable w1-gpio and w1-therm
* Know the ID of the probe
* Read the probe’s output– if the CRC is ‘YES’, get the temperature data and convert it to Celsius from milicelsius
* Print the temperature in Celsius to the terminal

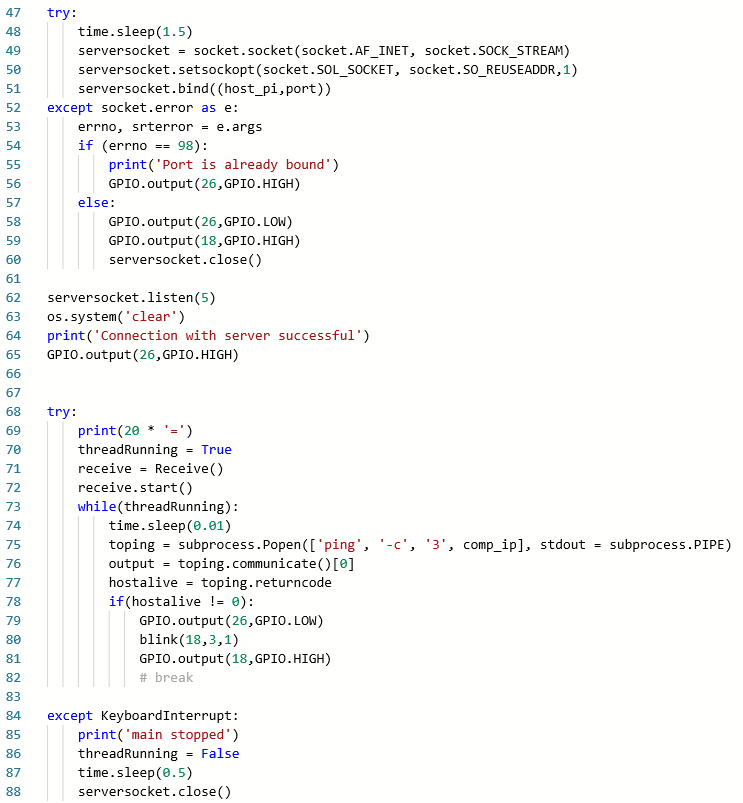
*Server Code*



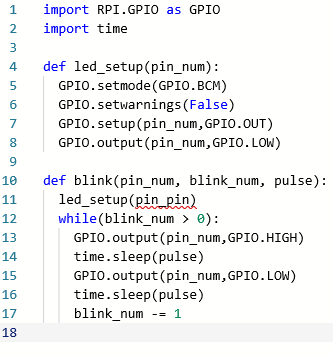
* Import the necessary modules
* Define the connection parameters (host IP and port number), create a socket, then listen for connections
* Using a thread, when there is a connection, run the temperature script and read its output from the terminal
* Format the message and send it to the computer
* Handle possible errors
* Close connection

Additional Comments

* Lines 53 – 60: handling any issues with connection
  + Error #98 means a connection is already established and can’t make a new one
  + Error #99 means that a connection cannot be made



* Line 75: checking if Ethernet is connected by pinging IP of computer

*LED Code LED.py*

* Import the necessary modules
* Set up the GPIO pin based on a pin number that is defined by the user
* Create a blink function that takes in the GPIO pin number, how many times the LED blinks, and pulse length

*Running a Python script on startup of a Raspberry Pi*

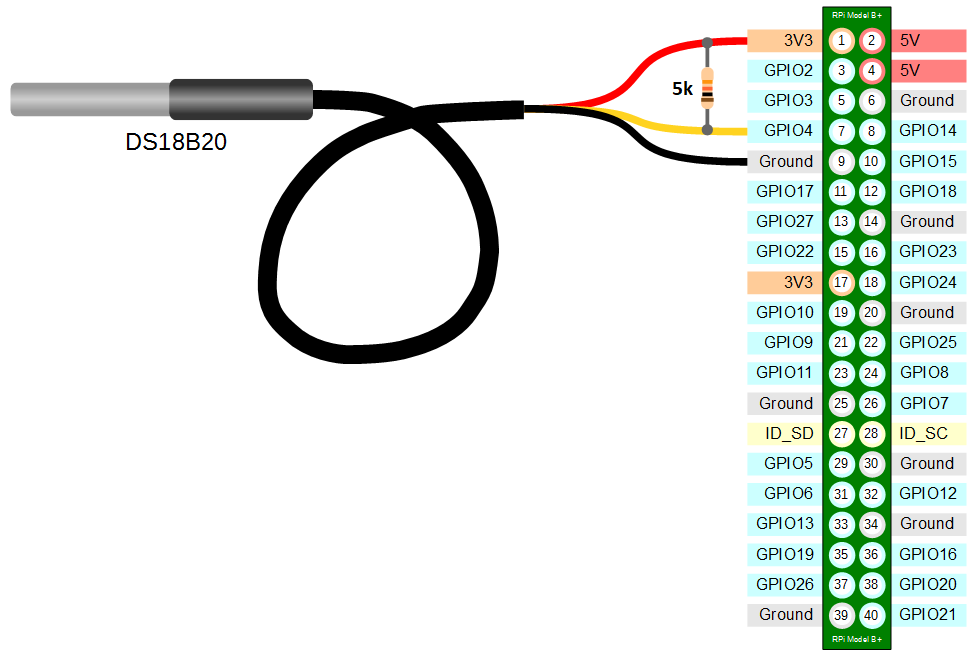
1. Know the file path for the server script script
2. Run sudo nano /home/pi/.bashrc in a terminal window
3. Run cd /home/pi/.config/lxsession/LXDE-pi
4. Run nano autostart and in the editor add @midori in a new line, followed by @ python nameofyourscript in a new line
5. Type ctrl-X, Y, then enter

*How to remotely login to Raspberry Pi*

1. Use PuTTY – type IP of Raspberry Pi and connect over Port 22. Make sure SSH is selected

**-OR-**

1. Open the command line and run ssh pi@IPofRaspberryPi
2. When prompted, enter raspberry as the password
   1. Note: if the login has been changed, use newusername@IPofRaspberryPI



**Hardware Design**

Temperature Probe

* Red wire (VDD) goes to any 3V3 port on the Pi
* Yellow wire (Data) goes to any GPIO pin 4
* Black wire (Ground) goes to any GND port
* A 5k ohm resistor is placed between the GPIO pin and 3V3 pin

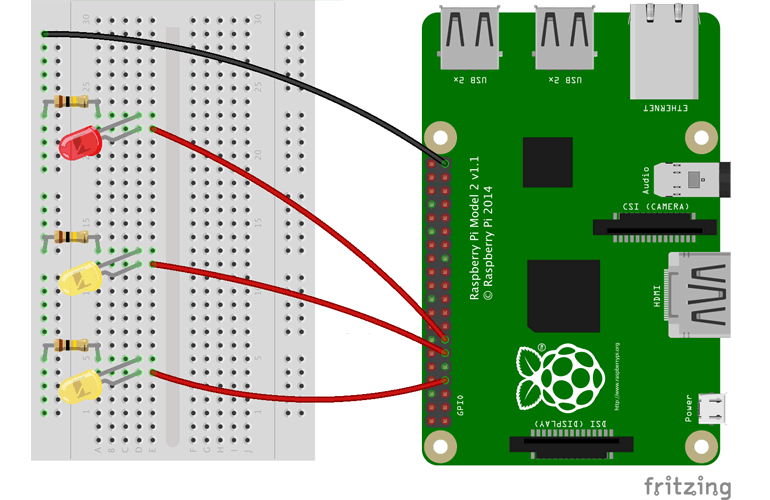
*Debugging Probe*

If the probe is giving error 8000, make sure that the probe is wired correctly. In the event that you need to switch out the probe, rewire the new probe, connect the Pi to a monitor or SSH into the Pi, then follow these steps (you can also go through these steps to check your existing probe’s connection):

1. Open a terminal window
2. Run cd /sys/bus/w1/devices and then ls
3. You may see multiple things pop up but look for a number that is similar to 28-030297944f2d
   1. If you have not changed out the probe, this might be the exact string you see
   2. If you have changed the probe, take note of the new id and change the temperature sensor code (line 30) to the new id
4. Next run cd xx, where xx is the probe’s id you found
5. Run cat w1\_slave and observe the output

If at any time one of these steps failed (get an error like “no such directory”), double check how the probe is wired. If the last step gives you a CRC value of NO or all values turn up 0, it is likely also an error with how the probe is screwed in.

LEDs

* Connect one end of the resistor (100-200 ohms) into the GPIO pin on the Pi and the other into the through holes on the side of the breakout board
  + Red LED is in GPIO 18
  + Green LED is in GPIO 26
  + Orange LED is in GPIO 16
* Using the “grabber” wires, clip onto the side of the resistor in the through hole and clip the other side of the wire onto the cathode pin of the LED (longer leg)
* Plug in the anode pin of the LED into GND on the Pi

*LED Codes*

|  |  |  |
| --- | --- | --- |
| LED Color | Indicator | Meaning |
| Green | On | Connection with Pi Ready |
| Green | Blink Once | Request received |
| Red | On | Ethernet was unplugged or connection failed |
| Red | Blinks | Ethernet became unplugged – request timed out |
| Orange | Blink 5 Times (short) | Temperature probe is not connected (error 8000) |
| Orange | Blink 5 Times (long) | Error 7999 |

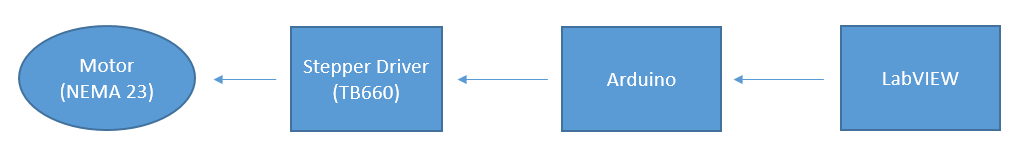
Notes:

If the Green LED is on and then the Red LED flashes after a short amount of time, it is likely an issue with the network’s firewall. Go into the computer settings > Network & Internet > Ethernet > Network 3. Check that the Network profile is set to Private.

If the Red LED is on when Arduino is first plugged in, try unplugging and replugging Ethernet cord. If that does not help, SSH into Pi and run python pingtest.py.

**Motion Control Documentation**

**Block Diagram**



* The stepper driver is set to 200 pulse/rev, which means that the motor steps 1.8 degrees with each pulse. The current is also limited to 1 amp
* An Arduino Uno is used and a shield is installed for access to the GPIO pins
* The LINX toolkit on LabVIEW is installed to interface with the Arduino

**System Description**

VI Inputs and Outputs

Rot/Lin

Freq

Cluster

MotionControl VI

Dir

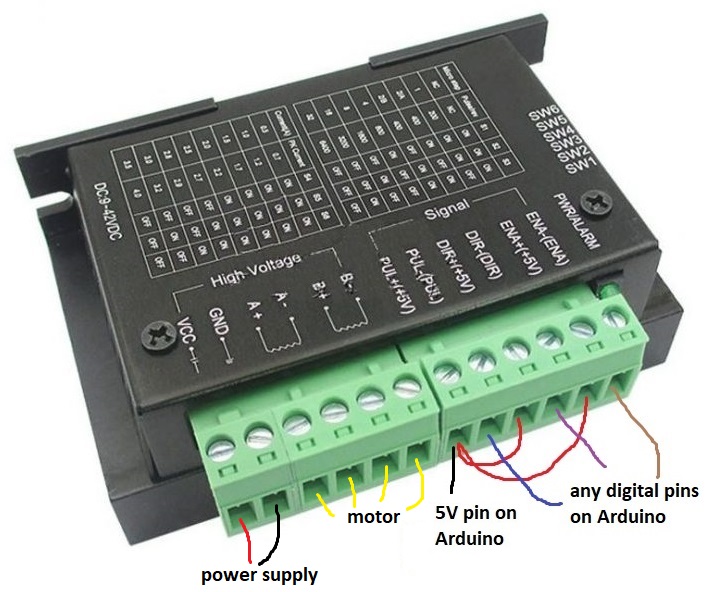
Error Out

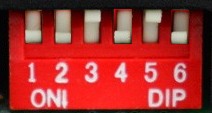
Dist

Error In

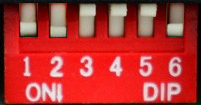
* Rot/Lin specifies if the motion is rotational or linear
* Freq is the frequency of the pulse – generally, the rotational motor should be set to 1000 Hz and the linear track should be set to 650 Hz
* The cluster contains all the system parameters, including the COM port of the Arduino.
* Direction
  + For rotational movement: True is counterclockwise and False is clockwise
  + For linear movement: True is up and False is down
* Dist is how many degrees to rotate (if rotational) or how many millimeters to travel (if linear)

Driver Wiring Diagram

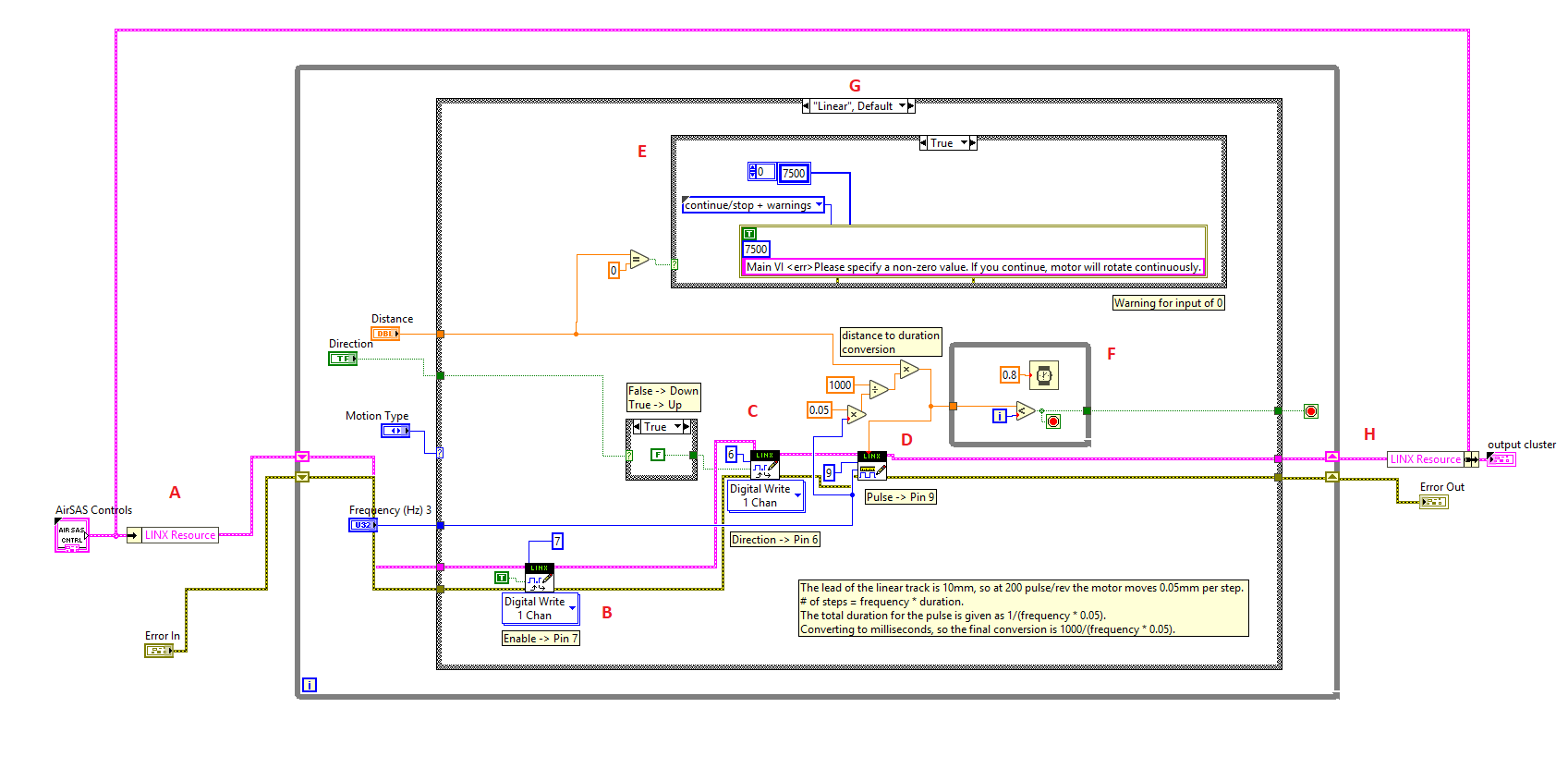


* Use three GPIO pins for the turntable and another three GPIO pins for the linear track.
  + In our case, GPIO Pins for the linear track are 6,7,9 and the GPIO Pins for the turntable are 10,12,13.
* A wire must also go into the Arduino 5V pin for each motor, and this feeds into the PUL+ input on the driver. Connect wires from PUL+ to ENA+ and DIR+.
* ****The other GPIO wires go into PUL-, DIR-, and ENA -, and keep track of which GPIO pin goes to what input on the driver.

turntable

* The pins on the driver for the turntable should be ON ON OFF ON OFF ON
* ****The pins on the driver for the track should be ON ON OFF OFF OFF ON

track

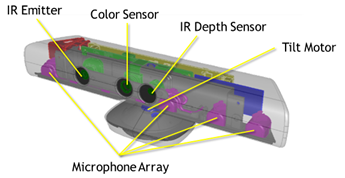
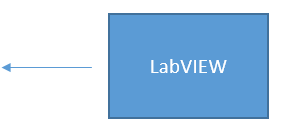
**LabVIEW Block Diagram**

1. Passing a LINX Resource which contains the serial connection with Arduino via a serial port
2. Enabling the motor to step. 7 is the GPIO pin ENA- is connected to for the linear track
3. Controlling the direction of movement. G is the GPIO pin DIR- is connected to for the linear track
4. Sending the square pulse to step the motor. 9 is the GPIO pin PUL- is connected to for the track.
   1. In order to step the proper distance on the linear track, a conversion is necessary. The lead of the linear track is 10mm, so at 200 pulse/rev the motor moves 0.05mm per step. # of steps = frequency \* duration. Given that the motor moves 0.05mm/step, the total duration for the pulse is given as 1/(frequency \* 0.05). The square wave write function is given in milliseconds, so the final conversion is 1000/(frequency \* 0.05).
5. This is the custom error handler for error 7500. If the user inputs a distance that is 0, this will cause the motors to rotate continuously. This error displays a custom error message – “Please specify a non-zero value. If you continue, motor will rotate continuously” along with the option to stop the VI or continue. Stopping the VI before this happens is strongly recommended, as cutting power to the Arduino is the only way to stop the motor once it begins to rotate continuously.
6. Creating a delay such that the VI stops running after the motion is fully completed
7. Case structures to handle if it is rotational or linear movement
8. Return LINX Resource

Note that you must load the LINX firmware onto the Arduino before running the VI for the first time. To do this, in the Motion VI, go to Tools> MakerHub > LINX > LINX Firmware Wizard. Select Arduino Uno and the correct COM port.

**Kinect Control Documentation**

**Block Diagram**



* Stream video from the color and depth sensor to LabVIEW
* The Kinesthesia Toolkit on LabVIEW is installed to interface with the Kinect

**System Description**

* Kinect for Windows v1
* Windows Kinect SDK v1.0 (installs necessary drivers)
* The Kinect must be plugged into the computer via a USB-3 Port (inside port is blue)

**imgToHDF5.vi**

This VI converts a 2D picture into an image in HDF5 format. This is a sub-VI.

VI Inputs and Outputs

imgToHDF5

Path

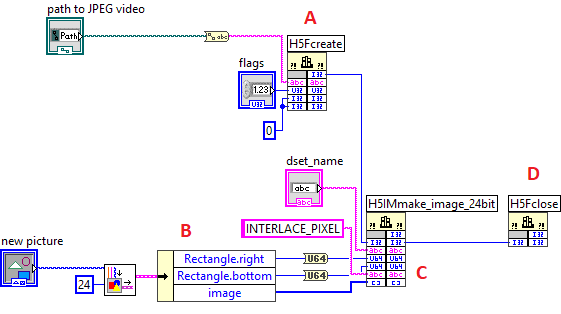
image

flags

name

* Path is the file path for the HDF5 file
* Name is the name of the dataset that is created and written to for the image
* Flags are overwrite codes for the image file path
* Image is the 2D picture

Block Diagram



1. Creation of HDF5 File with path specified in GUI through direct h5.dll function call
2. Getting image information from 2D picture
3. Creating a 24-bit image; creates and writes a dataset named dset\_name attached to the desired file containing the image information
4. Closing the HDF5 file

**PixelDistance.vi**

This VI allows users to find how many pixels are in a one centimeter square on the turntable. The turntable is of a known dimension (1 foot by 1 foot).

VI Inputs and Outputs

2D Picture

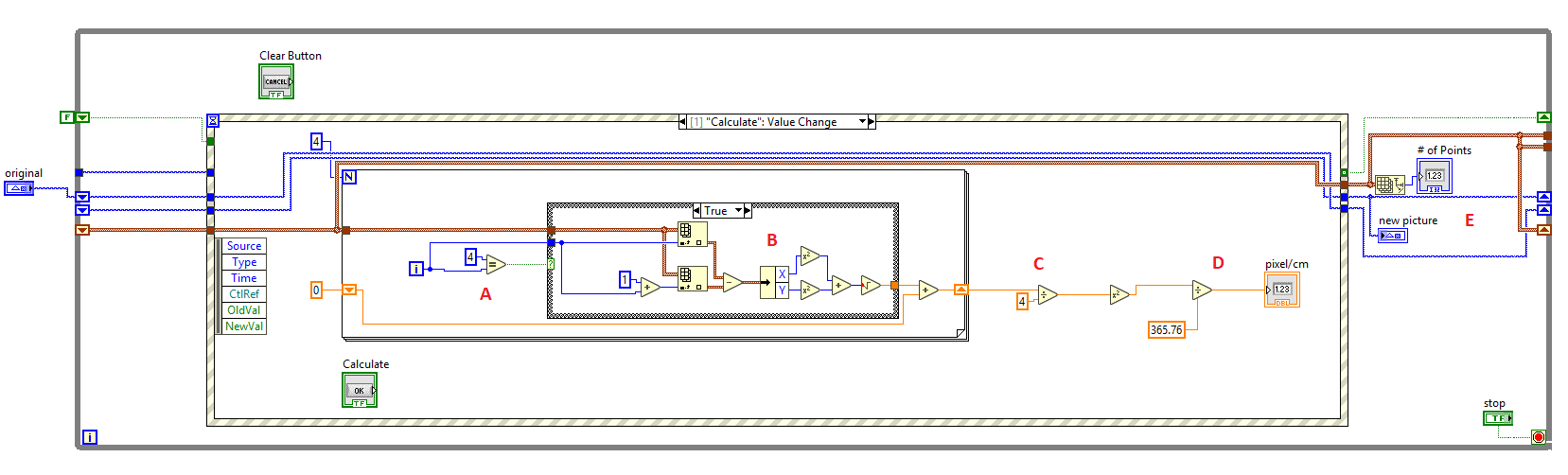
imgToHDF5

* The input the VI is simply the 2D picture at the time “OK” was pressed in KinectVI.vi
* The output is the display on the screen of the number of pixels per 1cm x 1cm area

Instructions for Use

* Select four points at the corners of the turntable in a counter-clockwise or clockwise order
  + Do not select points out of order
* The user can drag to create lines or create discrete point
* The number of points is displayed on the bottom in case the user accidentally creates a point and is unaware of it
* Press “CLEAR” to clear all points made
* Once satisfied with the four points, press “OK” to calculate the pixel per centimeter count

Block Diagram

This is the main event case – other cases deal with drawing on the image

1. Checking that four points have been plotted
2. Calculating the distance between the current plotted point and the point before it
3. Calculating average of all four lengths (since should be square, so equal side lengths), then squaring side length
4. Dividing by area of turntable in cm and displaying number
5. Displaying number of points

**KinectVi.vi**

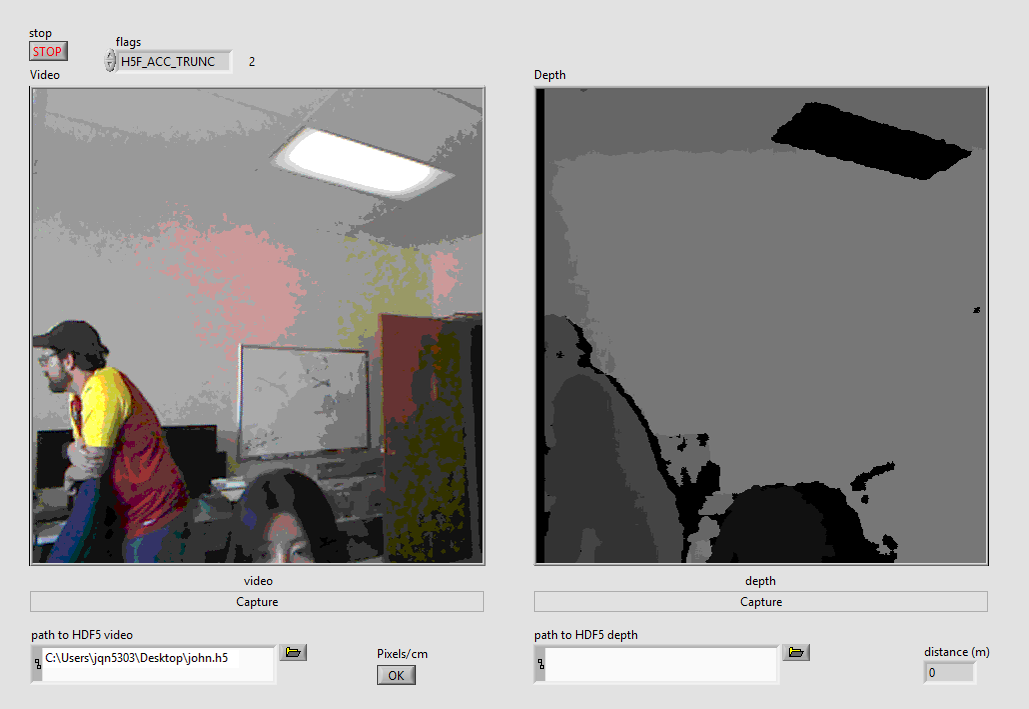
This VI streams the color video and depth map to the user. The front panel of the VI allows the user to capture images from these video streams, along with the file path where the images should be saved. Images are converted into HDF5 files following the HDF5 Image specification through the imgToHDF5 VI.

VI Inputs and Outputs

* The inputs to the VI are the file paths for the HDF5 files and the flag
  + Flag 2, “H5F\_ACC\_TRUNC”, truncates file. If the file already exists, it erases all data previously stored in the file
  + Flag 4, ““H5F\_ACC\_EXCL”, fails to save the file if it already exists.
* The only outputs are the saved HDF5 files from any images captured.

Note: When loading this VI, you might be asked to located the Microsoft.Kinect.dll. Navigate to Local disk in the file manage r> Windows > MICROSOFT.NET > assembly > GAC\_MSIL > Microsoft.Kinect > v4.0\_1.0.0.0\_\_31bf3856ad364e35 > Microsoft.Kinect.dll. Click on it and then press OK. There might be some warnings, however these can be ignored.

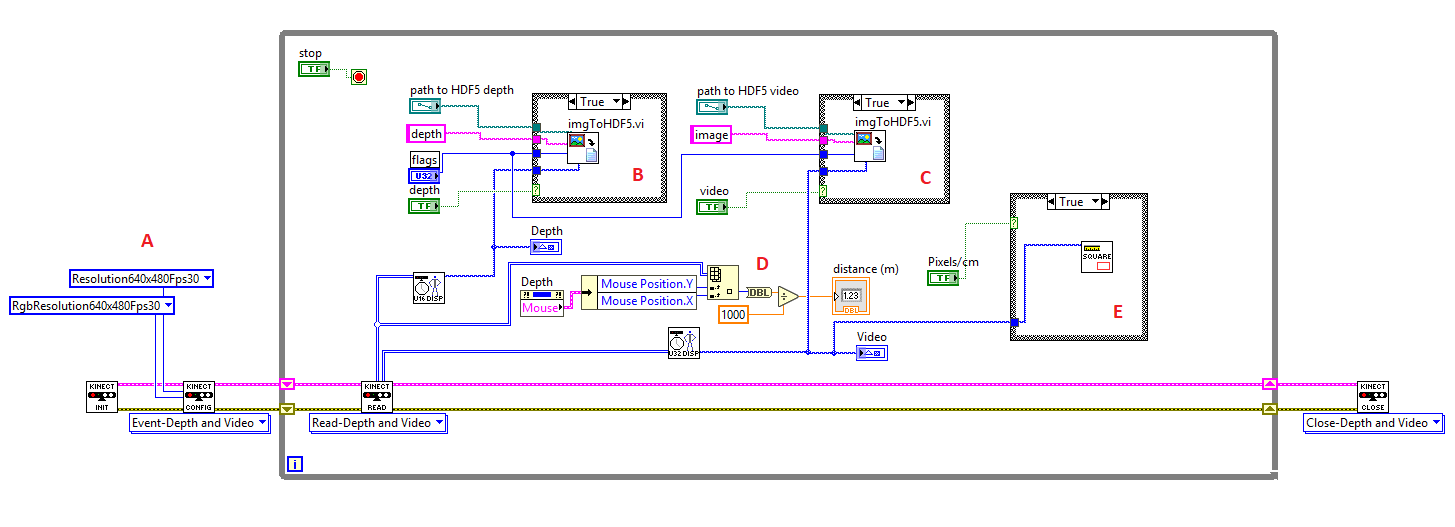
Instructions for Use



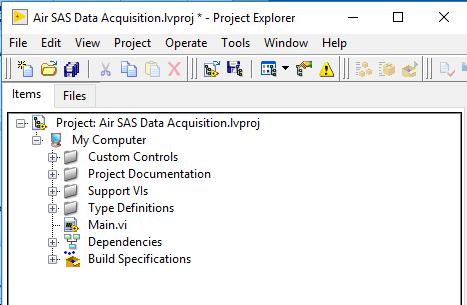
* Check that the Kinect is properly connected – a green light should light up on the front of the Kinect
* Specify the path for HDF5 video and the flags before starting the VI
* Press the “Capture” button to save an image from the desired video stream
* Users can hover over the depth map and find the estimated depth in the bottom right corner
  + Distances are not accurate if the object is closer than 2 feet in front of the Kinect or 20 feet away from the Kinect (0.7–6 m)
* Users can also press “OK” to acquire the amount of pixels per centimeter on the turntable.

Block Diagram

1. Specifying resolution of video and depth map streams



1. Converting captured depth map frame into HDF5
2. Converting captured video frame into HDF5
3. Getting distance from mouse hovering over depth map
4. Using current frame of video and opening VI to get pixel to distance ratio



**Air SAS Data Acquisition**

**Steps to Running Data Acquisition**

1. Make sure all systems are plugged in and turned on
2. Open the LabVIEW project titled “Air SAS Data Acquiosition.lvproj”. A window like the one shown to the right should appear
3. Open “Main.vi” and run the program and wait for the Display to show “System Initialized”
   1. If the connection to the Arduino was not closed properly before running Main, Error 5003 will occur. Click “continue” and run the VI again
4. For manual control of the waveform capture, click the “Waveform Capture” button once the system is initialized (the button will be greyed out before this)
5. For manual control of the motors, click the “Motion Control” button once the system is initialized. This should never be necessary unless the system crashes or needs to be stopped for some reason, and the program never re-initialized the motors.
6. To run the data acquisition program, click the “SAS Data Acquisition” button once the system is initialized. This should cause the RunSAS.vi window to open if it was not open already.
   1. If this gives an error, it is because the file path for the data was not specified in RunSAS.vi. To avoid this, add a file path to the VI before running Main.vi. To do this, access RunSAS.vi in the Support Vis folder in the LabVIEW project window.

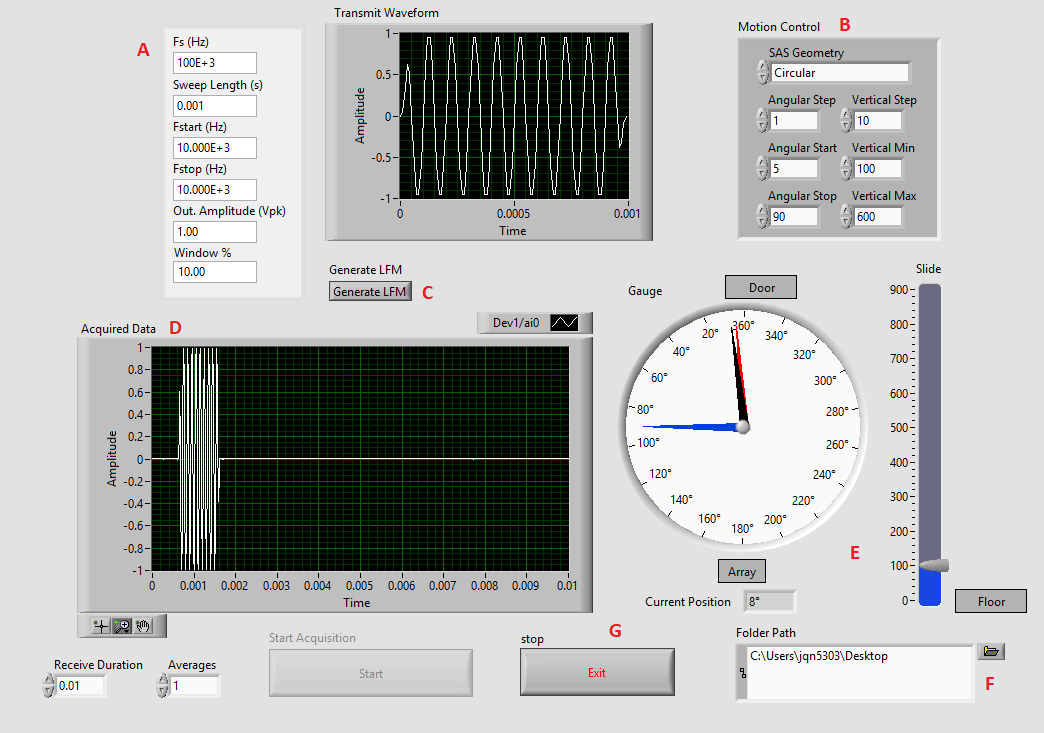
**newRunSAS.vi**

This VI runs all aspects of data collection, integrating the motion control, data acquisition, and writing to CSV for three different types of SAS geometry (circular, cylindrical, and spiral).

VI Inputs and Outputs

* The inputs to the VI are the cluster of system parameters and any errors
* Outputs are the cluster, any errors, and CSV files with the data

Instructions for Use



1. Specify the transmit waveform parameters
2. Specify the geometry of the SAS as well as motion control parameters
3. Generate the waveform, which will be shown in the graph above – This must be pressed before pressing “Start”
4. Shows the waveform data that was acquired
5. Indicators to show current position of turntable and linear track
6. Folder path of where to save data – this must be populated before running Main
7. To stop the VI – the VI will not close, however by pressing “Exit” on Main the entire program will stop

Notes: Any time when editing a value field, make sure to press enter or the checkmark in the top left corner to save the value, otherwise it will revert back to its previous value

**SubVIs for newRunSAS.vi**

1. newTemp is the temperature data function
2. CoordinateSave is saving system coordinates to CSV
3. InitTrack is function that moves linear track to min vertical height
4. NumSteps calculates the number of steps of the motor
5. Re-Init reinitializes the linear track and turntable
6. SysParamsSave save system parameters