Software Resource Document

Characterization Software

Installation materials / libraries / environment:

Raspberry Pi 3 - can be purchased on amazon

Raspbian 3 OS - (https://www.raspberrypi.org/downloads/), Download the image file and mount it on a >= 32GB microSD using your preferred disk imaging software (WinDisk32 on Windows). This microSD will be the disk drive for the Raspberry Pi. Be sure to turn off Raspberry Pi before removing the power cable to prevent corrupting the disk image (and losing everything!).

Python 3.x - Should be installed with Raspbian distribution, but can be checked by running <python --version>. If the version is 2.x, run <apt-get install python3.3> to install the newer version to the Pi. Check that the new install is 3.x by running <python --version> yet again.

OpenCV 3 and Requirements - As outlined in: http://www.pyimagesearch.com/2016/04/18/install-guide-raspberry-pi-3-raspbian-jessie-opencv-3/

Argparse and argcomplete libraries for Python - <python pip install argparse> on the command line to install the python package.

VLC - <apt-get install vlc> on the command line. The installation can be verified by opening any video file with <vlc [video file]>

Modules and Files:

To acquire the characterization software clone the git repo in a desired folder with <git clone https://github.com/CIDARLAB/Vulcan.git>. All of characterization will be located in /Droplet_Characterization/Characterization_Dev/ with the following modules:

VulcanParseAndFormatting -

A module folder that contains the modules to handle the I/O formatting and interpretation to the algorithm to configure data input defaults and output file format and location.

CharacterizationInputParsing -
This module uses the argparse library to take command line arguments for python files. The constructor defines the command line flag, argument, and argument description for each available argument. There are currently no defined getter / setters but all the member variables and methods are public. Pythons that need to use this library can access command line arguments by calling the object and passing in a string key to get the specified argument passed (<object>.parsing['variable'])

CharacterizationOutputFormatting -
The module takes argument in its constructor to define formatting parameters for data. Given these params and the data in list format, the main python file can call functions in the module to format the data in whatever fashion to a defined path. Currently, only csv is defined but can be expanded to other formats by adding additional methods to the module to be called.

DropletCharacterizationXMajor.py

The python file with the current, most up to date implementation of characterization. As the name suggests, it characterizes droplets moving in the X direction with a reference. It takes command line arguments as defined in CharacterizationInputParsing. After extracting the command line flags, the program goes through the video file frame-by-frame and runs through the filter and computer vision parsing for each frame.

To actually detect droplets (which have a sharp contrast with the background of the chip), the OpenCV library is used to conduct computer vision analysis on the video / feed. The basic algorithm begins with blurring the image of soften the edges in each frame. A binary threshold is then applied to the frame to clearly differentiate the droplets from the background, leaving the droplets white against a black background for maximum contrast. A contour search is then applied to the frame to detect blobs, which will be the droplets. We can characterize droplets by performing operations on contours, which allow us to grab items like area and pixel RGB.

To track individual droplets in a moving line, the concept of droplet eclipses and detection zones were introduced. A detection zone is a line running parallel to the droplet channel (therefore perpendicular to droplet movement) to track droplets as they cross the line. This act of crossing is called a droplet eclipse. To differentiate droplets from each other, there must be a gap between eclipses,

defined by at least one frame that is not an eclipse. To capture data about droplets, the time spent in an eclipse, FPS of the video, time between eclipses, and the relative velocities of consecutive droplets are used in calculations. Data is collected for every droplet and is placed in a list to be fed into CharacterizationOutputFormatting to be output a file.

Important

There are two lines of detection, one running along the X-Axis, and one running along the Y-Axis. The Y line (Y_DETECTION_BORDER) is used to align the reference point. The X line (X_DETECTION_BORDER) is to be placed above the channel, but below the reference. This prevents the reference from being counted as a droplet in every frame. The values for the axis lines can be changed by the variable mentioned above (X/Y_DETECTION_BORDER). The coordinate plane is a CS-style coordinate plane, where the origin is on the top left.

SerialComms.py

This python file handles the serial interface between the Pi and the Peripheral Manager (or any software). The program listens for incoming data on the UART port and makes decisions based on input. To handle live I/O between the serial port and inter-process communication, callbacks for data were not possible to use. Instead, the serial port is read for 1 second every iteration (data will not be dropped because it will be buffered via the UART protocol) and the time between reads will be realistically under 100ms. In normal operation, the program will spawn a child process with specific command-line run instructions that include run flags. Currently, the program is setup to run *DropletCharacterizationXMajor*, when <run live> is received on the port. This can be expanded to run any program by altering the run instructions given certain inputs. As the interface between the Peripheral Manager and modules is home-grown, it can be expanded to handle additional parameters and programs. Data is passed from the child process to the parent by keeping a pipe open and writing the result to a string at every loop iteration. At the end of the iteration, data is written out to the UART port. Upon receiving a kill command, the child process is terminated and a confirmation is sent through UART.

PCR Software

Installation materials / libraries / environment:

Arduino Uno - can be purchased online from Amazon.com

Arduino Software - can be downloaded from: https://www.arduino.cc

Arduino Library Install: https://www.arduino.cc/en/guide/libraries
PID Library: http://playground.arduino.cc/Code/PIDLibrary
Thermocouple Reading Library: https://github.com/adafruit/MAX6675-library

PCR_Controller Arduino Project:

PCR_Controller.ino -

Upload this file to an Arduino as any normal project. It will expect commands through serial communication at a baud rate of 115200. Properly connected to a chip and a circuit (see hardware documentation), the Arduino will use a PID controller to vary the power through the circuit in the nichrome wire in order to heat or cool the chip.

With ‘T’ as the command for starting a time array, ‘H’ as a command for starting a temperature array, ‘S’ for the end of each array, ‘W’ as the warm-up condition, and ‘Q’ as quit inputting commands, the Arduino will properly take in the thermal cycle as arrays and implement them one the user has inputted valid arrays.

For example if the user inputs:‘W H 55 65 S T 10 20 S’, The Arduino will warm-up the chip first, heat it to 55 degC for 10 seconds, then heat it again to 65 degC

The code utilizes a state machine to determine when the Arduino should be listening to commands as well as when it should be executing the thermal cycles. Global variables keep track of the array and what position in the array the arduino is currently executing. The user can press Q to stop the thermal cycle at any point. When the temperature and time arrays have been completed executed, the Arduino will go back to waiting for the user to execute another thermal cycle.

Peripheral Manager Software

Installation materials / libraries / environment:

Node.js - Download here as per your operating system.

Files and Dependencies:

Files - To acquire the peripheral manager software clone the git repo in a desired with <git clone https://github.com/CIDARLAB/Neptune-Peripheral-Manager.git>.

Dependencies - Type <npm install> while in the peripheral manager folder which contains package.json.

app.js - The server is listening for connections on port 3000. It is possible to use the peripheral manager with a command line interface or via a GUI. The manager perform the following actions on receiving the corresponding command on the web-socket:

- Return list of all active ports and connections - 'get ports'
- Establish connection - 'Connect', port name
- Delete connection - 'Disconnect', port name
- Send Message to port - 'send message', [port name, message]
- Update reference name - 'update name', [port name, reference name]
- Create connection b/w modules - 'establish connection', [source, dest]
- Create virtual connection - 'virtual connection', file name
- Write to virtual connection - 'virtual message', [file name, message]

index.html - This file provides a fleshed out GUI to access all of the peripheral manager's functionality. It has methods to display information about PCR and Droplet characterization in an aesthetically pleasing way.

Usage:

1. Type <node app.js> while in the peripheral manager directory on your command line interface.
2. Connect to localhost:3000 on your browser to access the manager.
3. Connect peripherals and send messages as desired.

Liquid Flow Relations (LFR) Software

Installation materials / libraries / environment:

Java - Downloadable from: https://java.com/en/download/

Project Files:

muShroomMapper: Software tool which takes the user defined LFR and User Constraints File (UCF) and translates it to CIDAR-microfluidic-standard MINT code for Place and Route.

Main.java: Calls all other class files to translate LFR into MINT. This file is customized for use by either a command line user or a Neptune GUI user.

DebugMain.java: Calls all other class files to translate LFR into MINT, then also calls the CIDAR Place and Route tool to compile the MINT into a physical blueprint. This aids debugging and is meant for developing and testing from the Netbeans (or any) IDE.

ParsedUCF.java: Reads in the user's JSON format UCF file, which describes all parameters the user desires for their microfluidic components. If the user has enabled PCR for their microfluidic chip on the UCF, the creation of the physical blueprint for it is made.

NetListTransition.java: Calls NetSynth (software tool described later) to parse the LFR (comprised of modified Verilog *assign* statements) and receives lists of channel (wire) and component (gate) objects. These lists are then informed with the information stored from the UCF, then turned into a graph.

CreateMint.java: Takes the informed graph of microfluidic components and their connection channels from NetListTransition. Checks for any multiple output definitions and splits output from component into multiple channesl if so. Uses informed graph and parameters defined in UCF to tranlsate chip design to MINT, the CIDAR standard for microfluidic design for Place and Route.

NetSynth: Software tool that converts Verilog assign statement code into a list of wires and gates created by Prashant Vaidyanathan of CIDAR lab. Shane

McCormack branched off of this tool to fit the needs of muShroomMapper, and parse LFR, modified Verlilog assign statements, instead. These edits included changing the ANTLR grammar and Walker, adding attributes to Gate and Wire object for compatibility with the microfluidic standard, and creating new types of wires (“channels”) and gates (“uF”) for use in the tool.