### 8/15 Notes

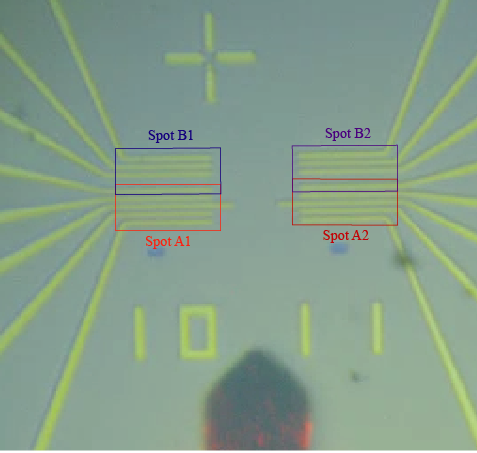
* Licensing to LabView’s computer vision package has expired
  + Need to implement ezAFM Automated Imaging with less dependency on image recognition tools
* To start off, we can have the user manually center the tip on the crosshairs and then automate the following scan for that specific pair of devices
* Once the tip is centered on the crosshair, the user may select to scan Spot A1, A2, B1, or B2
  + After each scan, the user would have to re-center the tip on the crosshairs
* We can make it easier on the user by implementing the following:
  + Move2Device button for Coarse Navigation
  + Fine adjustment for scanning the given spots A1, A2, B1, and B2 in case the resulting scans come out to be a bit off
* The majority of the testing will be in pinpointing the fine navigation of directing the tip from being on the crosshairs to one of the 4 defined spots
* For the future, when we are able to implement image recognition tools:
  + We would already have approximate instructions on how to move the tip after identifying the tip’s position with respect to the crosshair
* We might be able to selectively choose which electrodes to scan
  + By default, we may move to the 4 defined spots. From there, we can move vertically up to the electrode of interest (still starting on Region 1)
  + NOTE: Sample rotation would have to be minimized otherwise scanning region would be very inaccurate

### 8/15 Tasks

* Optimize time delays
* Calibrate speed to distance traveled ratio
* ~~Fine navigation~~
  + ~~Obtain coordinates from table (Ask Prof. Collins)~~
* ~~Figure out “Current Device” to “Target Device” travel function~~

### 8/23 Notes

* Since there is no LabView computer vision licenses, I resorted to using “ezAFM Scan Settings from WaferDevice” VI
* The scan “Spots” are defined below



* The coordinates are given by:
  + Spot A1 = (-73, 35) Spot A2 = (-73, -59)
  + Spot B1 = (-52, 35) Spot B2 = (-52, -59)
* Currently, I’ve just hardcoded a series of operations to perform once the user has centered the tip on the crosshairs
* We may be able to implement a feature to selectively choose which electrodes to scan by adding/subtracting to the x-coordinate

### 8/25 Tasks

* Need to implement global variables in order to keep track of status indicators
  + “Tip Approached?”, “Approaching”, “Retracting”, “Scanning”, “Current Device to Scan”
    - Unable to update the indicators with the correct status while performing the action. I think this might be because of the data flow issue of putting the indicators outside of the helper VI’s but updating their status within the VI’s, which may result in no change appearing since the indicator does not update during the helper VI’s operation
* Even with a reasonably square sample orientation, I still missed the ab electrode in trying to scan Spot A2 despite capturing it in Spot A1
  + I will include the angle transformation in the coordinate vectors to scan and try to scan Spot A2 again
  + This will also prove the precision of our theta value for fine navigation
    - In order to see if our Move2Region vector is able to keep up with the electrodes, we must have an angle of at least 2 degrees, otherwise the angle transformation has no effect, which suggests we should just use arbitrarily add/subtract a few microns to the vertical movement of Move2Region depending on the sign of the angle
* See if we can accurately move post-scan, so after scanning the 3 regions for a given spot, test to see if we’re able to move to the next spot and scan again accurately
  + If so, this means we’re able to take 6 scans reliably, capturing the entire device
* ~~Need to implement file saving/renaming system~~
  + ~~So far my “FileManagement” VI just takes the latest 3 files in the ezAFM Default Directory and copies/renames them into a new directory~~
  + ~~All the user has to do is input the new directory path~~
* Another long-term task would be to incorporate the WaferMap functionality where the user opens up the menu to choose specific electrodes and builds the list of connections
  + Prof Collin’s version of the WaferMap functionality seems very complicated, I might just make my own
* The purpose of allowing the user to select several connections is so that we build a list of connections and then generate a sequence of movements in order to automate the scanning process as much as possible
  + Essentially the idea would be something the lines of

1. Build a list of the user-selected connections; organized from 00 to 33 devices
2. Move to the crosshair region of the devices of interest
3. Prompt the user to center the tip on the crosshairs
4. Scan the regions of interest
5. Repeat Steps 2-4 until all connections have been scanned

### 8/25 Notes

* Cancelling the scan by hand would cause the LabView program to freeze
  + Wait I’m not sure about this anymore. I cancelled the scan in the first region and the LabView program knew to somehow retract and then went back to running like normal
    - Need to test again
    - After testing again, the automatic retract no longer occurs and the program freezes
* We can use the “Current Device to Scan” string indicator as a file name for the new renamed files into

### 8/26 Notes

* I realized that the ezCommander software includes a “measure angle” tool in the imaging tools tab.
  + We may be able to take a partial preliminary scan of any set of electrodes (256 pixel) and use said tool to measure the angle
  + This would take approximately 10 mins which may or may not be better than computing the angle through navigation

### 8/27 Notes

* It is possible to scan both Spot A and B sequentially for a given device
  + Haven’t tested for significant sample rotation yet
* In the process of implementing a “SelectConnections” VI
  + Able to list out individual binary connections but now need to create a filtering VI that simplifies the list of connections
  + Then need to create another VI that generates the sequence of movements corresponding to selected connections

### 8/28 Notes

* Just met with Prof. Collins, he advised me to switch up my LabView programming style so that we can take advantage of LabView’s graphical IDE and all its nuances
  + Need to adapt it to operate similarly to the already built ezAFM VI’s
* The next short-term step is to come up with a quick and dirty semi-automated ezAFM imaging VI that would prompt the user to adjust the crosshair when moving to a new set of devices
  + Then we can replace the user prompt with a function that uses image recognition to identify the crosshair and obtain its position
* Another future task is to troubleshoot the ezAFM’s built-in XY Motor positioning register
  + The registers will output random vectors when we send commands from LabView to the XY Motor
  + Once we get this to work reliably, we can then use the ezAFM Automated Imaging VI that Prof. Collins
  + If it turns out that this is an immovable object, then we can perhaps

### 8/29 Notes

* As Prof. Collins advised, I changed the coordinates on the ‘ezAFM Scan Settings from WaferDevice” VI which is where I got the coordinates for the Scan Spots
  + Now I would need to import these coordinates perhaps as global variables into my VI so that future readers can see where the information is coming from
    - The information would then be more centralized

### 8/31 Notes

* Now that LabView licenses have been renewed, we can implement the computer vision tools used to identify the crosshair, cantilever, and device numerals
* The ezAFM Automated Imaging 3.3 VI left off where the user selects the connections of interest and then hits the begin scanning button
  + The scanning feature still has yet to be implemented
* My rough idea is to use the computer vision tools to obtain the pixel coordinate of the crosshair and the cantilever
  + Then we can obtain those two coordinates and find the delta-x and delta-y in pixels; convert them to microns; and send a XY Vector command to the XY Motor
  + The issue is I’m not sure whether or not we’re able to obtain the coordinates of the center of the crosshairs and the cantilever
* Need to test how the image recognition works
  + So far the ezAFM Pattern Recognition VI outputs a boolean but instead we want pixel coordinates

### 9/2 Notes

* Adjusted ezAFM Practice 1.4 so that it no longer resembles C programming with all the controls/indicators (variables) on the side and the functions are hidden
  + Still need to show some more of the functions for the user to see
* Implemented the ezAFM Camera stream into the VI
  + Now need to test pattern recognition for current conditions
* Before testing pattern recognition, I need to create the ‘BuildSequence’ VI which takes in the string array ‘Connections’ and outputs a 1D array of essentially commands for the XY motor to move
  + I figured out how to obtain the list of string arrays from the ‘SimpleMap’ VI
  + For now, I will continue using my own version just for testing purposes since the ‘SimpleMap’ VI has some file path errors that may be bothersome

### 9/3 Notes

* After using the LabView example for Pattern Matching and also the ‘Pattern Match with Masking’ VI, I finally found a match with the crosshair template from the ezaFM Config folder and with the tip template
  + Now we can try to obtain the pixel coordinates of these matches, compute the pixel difference and convert that quantity into microns for the XY Motor to move so that the tip is centered on the crosshairs
  + SUCCESS! Obtaining the ‘Offset’ coordinates of the tip and the crosshair gives us the proper XY Vector to send into the XY Motor. Moving on medium speed got us to the crosshair fairly accurately
    - Now the only concern is how reliable this is since the matches may not be centered at the same spot
    - The next move is to create some sort of ‘MoveAndCheck’ VI which runs when the crosshair cannot be found.
* I dedicated a whole new VI called ‘Move2Device’ just for moving between devices. This new VI would take in a Current Device and Target Device.
  + I also created a ‘SetMotionDelay’ VI which takes in either RowColumn array for Move2Device or an XY Vector depending on the user and outputs an array for XY timer delay values.
  + It seems that moving between rows takes 12 seconds each and moving between columns take 25 seconds
* Similarly, I am continuing the ‘Move2Vector’ VI which just takes in the XY Vector and moves accordingly.
* I’ve kept pushing off the Optimize Time Delays task, it seems that I will finally complete this task since it is required for when the angle measurement causes the XY vector for Move2Device to deviate slightly too far off such that the timers might be wrong
  + So far I’ve gotten this
    - f[x = step size] = 10x + 1000 = milliseconds to wait
    - Pretty accurate for up to 3,000 steps without much delay after the first move
  + When it comes to below step size = 100, it takes less than a second to move, so I might just program a constant delay of 5 seconds for step size < 200
* As for fine navigation such as moving to Scan Spots, we can also implement a constant delay of 10 seconds rather than having to go through with the testing on slow mode

### 9/4 Notes

* Using image recognition for the crosshair may not be entirely necessary since most of the values are going to be the same. This means we might be able to replace the ‘Offset coordinates’ of the crosshair with a constant vector which would then be used to compute the XY Vector to send to the XY Motor
* I’ve revised the ‘SetMotionDelay’, ‘Move2Vector’, and ‘Move2Device’ VI’s
  + ‘SetTimeDelays’ takes in an XY Vector and a Motor Speed
    - For FAST mode: f[x = step size] = 10x + 1000 = milliseconds
    - For MEDIUM mode: f[x = step size] = 15x + 1000 = milliseconds
    - For SLOW MODE: I just used a constant delay of 30 seconds
  + I might just scratch Move2Device subVI and instead in the main VI I can just have 2 subVI’s, calling GetDeviceVector and passing the output into Move2Vector

### Automated Scanning Design (FlowChart)

* Have the user select the connections
* User then clicks ‘BEGIN SCAN’
* Take the string array Connections and create an array of strings (essentially instructions) that correspond to specific movement between devices and between scan spots
* For example
  + Let Connections include 00ab, 01ef, 20abef, 31ef
  + The string array of directions would resemble something like:
    - Assuming the tip is within the region of 00 | 01, then center tip on 00 | 01 crosshair
    - Move to Scan Spot A0 and take the scan
    - Re-center the tip on 00 | 01 crosshair
    - Move to Scan Spot B1 and take the scan
    - Re-center the tip on 00 | 01 crosshair
    - Move from 00 | 01 to 20 | 21 crosshair
    - Center tip on 20 | 21 crosshair
    - Move to Scan Spot A0 and take the scan
    - Re-center the tip on 20 | 21 crosshair
    - Move to Scan Spot B0 and take the scan
    - Re-center the tip on 20 | 21 crosshair
    - Move from 20 | 21 to 30 | 31 crosshair
    - Re-center the tip on 30 | 31 crosshair
    - Move to Scan Spot B1 and take the scan
* I created a ‘TakeScan’ VI which assumes the tip is centered on the crosshair. The input terminals for this VI are the Scan Spot and the Image Resolution
  + This VI moves to the scan spot and takes a complete scan of all 3 regions, and then saves all 3 images to a given directory
* This list suggests that we should add on a ‘Re-center tip on crosshair’ function to our Move2Device VI and our TakeScan VI
  + The Move2Device VI implementation requires repositioning and image recognition
  + The TakeScan VI implementation does not require any image recognition since it takes in a Scan Spot, it’ll simply move the ‘negative’ of the Scan Spot XY Vector

### 9/5 Notes

* When it comes to trying to reposition and identify the crosshair, we can’t seem to handle it within just 1 event because then the image acquisition of the ezAFM Camera would be locked on that single frame
  + Instead, we can devote a separate event which we can signal via Property Node set on Value(Signalling). This event would trigger from some other event, in which case, the while-loop would have finished once. This updates the ezAFM Camera image to the next frame.
* I created a quick and dirty repositioning pattern to check for the crosshairs
  + It just moves in a square pattern (Up, Right, Down, Left) and increases the step size after each loop
    - This would just be temporary until we can figure out a cleaner, more efficient strategy
* Now I can finalize the ‘BuildSequence’ VI

### 

### 9/6 Notes

* I think I just completed a working version of the ‘BuildSequence’ VI
* Now time to create a ‘InterpretSequence’ VI which takes in the Sequence and actually executes each element
* In building a way to interpret each step in the Sequence, we can activate them via Property Nodes set on Value (Signalling) and then transferring the information (from controls) as local variables
  + This would also mean we would have to make unique buttons, controls, and events for each function (Move2Device, TakeScan, CheckCrosshair, etc)
  + This makes it so that we can kill 2 birds 1 stone by also allowing the user to manually input information into the controls and using buttons to trigger the events

### 9/8 Notes

* To-Do List:
  + Decompose auxiliary VI’s and incorporate their components into the main VI
  + Test using ezCommander software to navigate the XY Motor and obtain the sample rotation
  + Create a quick and dirty version of Automated Imaging

### 9/9 Notes

* I edited the ‘BuildSequence’ VI so that it assumes the tip is hovering over 00 | 01 Crosshair
* I also edited ‘GetScanSpot’ VI so that it also returns an array of the leads to be scanned ie ‘abcde’ or ‘efgh’
* Good news, we were able to overcome the video pausing issue when it comes to using event structures by simply including another while-loop running in parallel with our main while-loop
  + Just because we’re able to run the video update in a separate while-loop only solves the issue of live-updates for the ezAFM Camera
  + We still cannot obtain new images for single event operations
    - Meaning we cannot implement the ‘Automatic Repositioning’ to identify the crosshair into just 1 event since the image would not update until the event finishes and the while-loop gets to reiterate
    - Nor can we implement ‘Automated Imaging’ in just 1 event since we need

### Automated Imaging Pseudocode

* To work around the event structure video update issue, I created 3 events for ‘Automatic Repositioning’ and identifying the crosshair
  + ‘Enable Automatic Repositioning’
    - Initializes the variables ‘Total Steps’, ‘Current Steps’, and ‘Direction’
  + ‘Search’, ‘SEARCH’
    - ‘Search’ is the indicator while ‘SEARCH’ is the button
      * I made an additional button so that the user is able to press it if they ever wish to automatically center the tip on the crosshair
    - Takes in the ezAFM Camera Image and runs the ‘IdentifyCrosshair’ VI
      * If found, then
        + we use ‘Move2Vector’ VI to center the tip on the crosshair on SLOW.
        + We trigger the ‘Enable Automatic Repositioning’ event to re-initialize all the variables.
        + If the ‘Automated Imaging’ is turned on, then we trigger the ‘Automated Imaging’ event to continue the next step
      * If not found, then
        + If ‘Automatic Repositioning’ is turned on, then we trigger the ‘Searching’ event to reposition the tip
  + ‘Searching’
    - Handles the operations with ‘Current Steps’, ‘Total Steps’, and ‘Direction’
    - If ‘Current Steps” does NOT equal ‘Total Steps’, then send a command to the XY Motor to move 20 steps in ‘Direction’ and wait 2 seconds
    - Trigger the ‘Search’ event to attempt to identify the crosshair
* Now for the ‘Automated Imaging’ algorithm, I made 2 events plus 1 more event to imitate the scanning procedure. In the end, we would have only 2 events because we’ll just trigger the ‘Scan’ event with the right variables set (‘Connection’, ‘Scan Spot’, etc)
  + ‘Initialize Automated Imaging’
    - Wires the ‘Connections’ string array into the ‘BuildSequence’ VI and wires the output into the ‘Sequence’ string array
    - Initializes the ‘Index’ to be -1
    - Sets ‘Enable Automatic Repositioning’ and ‘Automated Imaging’ to True
    - Triggers the ‘Automated Imaging’ event
  + ‘Automated Imaging’
    - Takes the ‘Sequence’ and uses ‘Index’ to get the specific step (element) in the Sequence
      * Checks whether the element contains the string ‘Scan Spot’ or ‘Crosshairs’
      * If ‘Scan Spot’ is found, then
        + get the Connection as a substring and wire it into the local variable ‘Connection to Scan’ string
        + Trigger the event ‘Test’ (for now, which would be replaced with the event ‘Scan’)
      * If ‘Crosshairs’ is found, then
        + Get the current and target crosshairs as substrings, wire them into the local variables
        + Trigger the ‘Move2Crosshair’ event
  + ‘Move2Crosshair’
    - Takes the ‘Current Crosshair’ and ‘Target Crosshair’ controls, wires them into the helper VI and then wires the output into the ‘Move2Vector’ VI
    - If ‘Enable Automatic Repositioning’ is true, then trigger the ‘Search’ event
      * Within the ‘Search’ event is a trigger to the ‘Automated Imaging’ event if ‘Automated Imaging’ is true
  + ‘Test’
    - Gets the ‘Connection to Scan’ and wires it into the ‘GetScanSpot’ VI which wires the output into a case structure
      * The case structure wires out an XY vector depending on the Scan Spot
    - The vector is wired into ‘Move2Vector’ and the negative of the vector is wired again into ‘Move2Vector’
      * The purpose is just to simulate fine navigation
    - Once the tip has returned to the crosshairs
      * Turn off ‘Scanning’ and ‘Test’ indicators
      * Trigger the ‘Automated Imaging’ event to move to the next step in the Sequence
* The weird thing is that our program bumps into errors involving the XY Motor
  + I think the inconsistency is when we trigger events simultaneously without waiting for an event of finish properly
  + The XY Motor would travel way too far off from its target area

### 9/11

* The XY Motor inconsistency is still occurring every now and then
  + Seems to happen only for the ‘up’ direction (6)
  + Not sure how to troubleshoot this issue
* I created my own Crosshair and Cantilever Templates for the pattern recognition VI
  + They work rather effectively, but still need to test
  + The biggest issue when it comes to testing is being unable to identify the crosshair due to either inefficient ‘Automatic Repositioning’ or some random error with the XY Motor being inconsistent
* I bumped into the issue of not being able to update status indicators live during events and operations since I update them inside my helper VI’s and the indicator is in the main VI
  + Instead, in the helper VI’s, I can trigger the global variable indicators and then in the same while-loop that I use to update the ezAFM Camera video, I can also update the indicators
    - Will need to test

### To-Do List

* Incorporate helper VI’s into the main VI
  + ‘ComputeAngle’ VI
  + ‘FileManagement’ VI
* Determine how to replace certain helper VI’s with the VI’s already built
  + For example, we can replace our ‘AutoLand&Approach’ and ‘Retract’ VI’s with the ‘ezAFM Control’ VI and also the ‘Auto-Retract with Pattern Recognition’ VI
* Determine if our current quick-n-dirty version of Automated Imaging is effective
  + Currently testing on PCB18-DG with some sample rotation
  + Now need to test with little sample rotation
* Create controls for the user to input the wafer type and die name
* File Management
  + Currently the ‘FileManagement’ VI just saves the last 3 scanned images from the ezAFM Default directory and renames/copies them into the Target directory

### 9/17 Notes