SE423 Laboratory Assignment 7 (Two Week Lab)
Robot Vision

Goals for this Lab Assignment:

1. Start using the Final Project's starter code. This code uses the motion tracking system, OptiTrack, to help the robot keep track of its Pose. This code also allows the robot to travel from one X, Y position to another X, Y position.
2. Learn to use the OpenMV camera. First program a simple blob detect algorithm. Then use the OpenMV's find_blobs function to find multiple blobs of a certain color.
3. Use the camera to find the distance a golf ball is away from the robot assuming that the golf ball is on the floor.
4. Add a state to the given code's state machine that has the robot follow a colored golf ball when it is seen close to the robot. When the robot sees the golf ball large enough it knows that the ball is close so start following. When the golf ball disappears from the field of view of the camera, go back to X, Y point following.

Prelab:

1. If you have not done so already, finish adding gridlines to your LABVIEW application to indicate the tiles in the robot's arena. This was assigned at the end of Lab 6.

Laboratory Exercise

Exercise 1: Using the FinalProject Starter Project and XY Control.

For this lab we are going to start with a starter project called "FinalProjectStarter". Make sure you have used Git to get all the "Course File Updates". Once you have your repository up to date, import the FinalProjectStarter project into your CCS workspace. Use the hammer icon to make sure you can build this project. Also you will probably want to switch the build to "Flash" so that your program is ready to run when you reset the Red Board.

Let's take a look at this code created with the FinalProjectStarter. You will find, that for the most part it includes all the code you have developed in the previous labs. I moved much of the code into a library file in order to reduce the amount of code in the main C file. Open SE423Lib.h and SE423Lib.c to find all the functions you had created in previous labs.

Also notice there are some new source files MatrixMath.c (Matrix functions for the Kalman filter) OptiTrack.c (Motion tracking system support functions) and user_xy.c (Control code to command the robot to X,Y positions.)

Finally take a look at the main C-file. Note, this code implements the Kalman filter to merge the OptiTrack (Motion Capture system) data with the Dead Reckoned ROBOTps position. For now just take it on faith that this algorithm works. I will be explaining how the Kalman filter works towards the end of the semester.

Find in main() the while(1) loop and notice how this code uses the pushbuttons to allow you to print more messages to the text LCD screen. When we run the code below, make sure to push the four buttons to see the different messages printed.

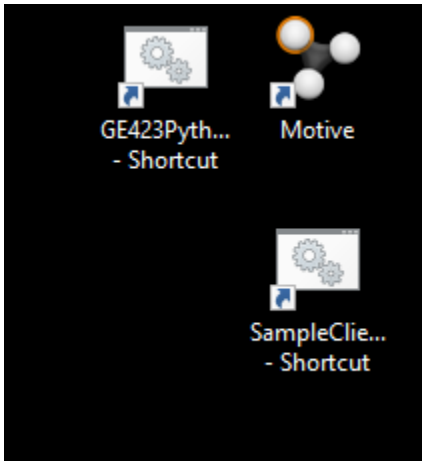
The flow of the code is the same as Lab 6:

- CPU Timer1 is called every 100ms and commands the LADAR to collect its 228 distance readings. 100ms is the fastest rate the LADAR can be commanded to produce readings. Also remember that distance index 0 and index 227 should not be used as they are blocked by the handle rails of the robot.
- ADCC is triggered by EPWM4 every 1ms. ADCC reads the four readings of the LPR510 rate gyro sensor. Currently we are not using this sensor data but I left it here just in case it becomes useful in the final project.
- When ADCC's four channels have converted, ADCC_ISR is called. Here the ADCC four channels are read. In addition in ADCC_ISR the SPIB communication with the MPU-9250 is initiated by writing the correct 8 16bit words over SPIB along with pulling MPU-9250 SS (GPIO66) low to select the MPU-9250.

- Once 8, 16 bit, words have been received over SPIB the SPIB_ISR is called. Here we read the IMU raw data and post SWI1_HighestPriority.
- When priority permits, SWI1_HighestPriority is called. Inside SWI1 will be where you develop most of you F28379D code for Lab 7 and the Final Project. Taking a look:
    - The IMU raw data is scaled to acceleration in g and rates in degrees/second.
    - Then the sensor zero values are found by the first 2 seconds doing nothing, the next 2 seconds summing up 2000 readings, at 4 seconds dividing the sum by 2000 to find the average zero and finally after four seconds subtract the zero value from the sensor reading.
    - For the next 2000 samples (2 seconds) a new feature is added and that is finding an average value for integral drift. This is a factor that can be subtracted for the gyro integral to help even more with the drift of the gyro's integral.
    - Finally after 6 seconds all zeroing has finished and the control code can be run.
    - Calculate gyro bearing
    - Calculate wheel velocity
    - Check to see if there is any new data from LinuxCMDapp, OpenMV camera data, LabVIEW data, or new data from OptiTrack (Motion capture data).
    - Perform Dead Reckoning and Kalman Filter data fusion of the motion tracking data. For now just know that this works to find the best value for ROBOTps.x, ROBOTps.y, ROBOTps.angle. I will be lecturing on this towards the end of the semester.
    - Then here after ROBOTps is found, is where you will be developing most of your code for Lab 7 and the final project. The function xy_control is called to command the robot to a new X,Y coordinate. Then the control state machine, switch case statement is used to transition between the many states the robot can have. In each of these states you are making the decision of what "vref" and "turn" are equal to, to make the robot perform correctly. Right now there are only three states: 1: commanding the robot to an X, Y point in the course, 10: Wall Following if an obstacle gets in the way of the robot. Currently the robot walls follows for a minimum of 5 seconds then tries go back to driving towards an X, Y point. 20: Follow at Orange or Purple golf ball. There is no code here currently and neither state 1 or state 10 transition to this state. You will be developing this code later in this lab.
    - After the state machine switch case statement, notice that PIcontrol() is called. This function implements the PI speed control of lab 5. Your code MUST call this PIcontrol() function every time SWI1_HighestPriority is called (every 1ms) for your wheel motors to run correctly.
- Just as in Lab 6 SWI2_MiddlePriority() processes the LADAR data.
- SWI3_LowestPriority() is currently not used and you can find a purpose for it in the final project.

Now you are ready to flash your F28379D board with the code and see how the robot is controlled with this starter code. Before you can run this code your instructor needs to show you how to setup and start the motion tracking system called the OptiTrack system. Steps below are to remind you:

1. Log into the Motion Tracking PC with username "dan5" and password "f33dback5".



2. Run the Application "Motive". Icon in top right corner of left monitor.
    a. Inside Motive a Pop Up window will display and select "Open Existing Project" and select the most current project that starts with "SE423…." To give you some more space move the Motive application to the right monitor.
    b. Then double click the "SampleClient –Shortcut" icon, also in the top right corner. Wait for this terminal to start displaying a bunch of data really fast.
    c. Then finally double click the "GE423Python – Shortcut" icon, also in the top right corner. Wait for this terminal to start displaying the coordinates of the robots it sees in the course.
    d. Now you are ready to run your robot. The Raspberry Pi application "serial_COMandOpti" receives the motion tracking data and sends it to the F28379D board.

Once OptiTrack is running, you can try out this code.

- Flash this starter code to your F28379D board.
- Next, to receive the motion tracking data, you will need to run "serial_COMandOpti" on your Raspberry PI4. Pull up a ssh or putty session and in your PI4 folder run serial_COMandOpti.
- In another ssh or putty session run the LVCOMApp so you LABVIEW program can draw where your robot is located in the course.
- Run your LabVIEW program from Lab 6 (with grid lines).
- Now place your robot anywhere in the course and press its yellow reset button. Wait the eight seconds for the robot to initialize its sensors and the flip on the motor amplifier switch to start the robot. NOTE: you should notice that the blue LED on the red board stops blinking when the sensors have found their zero values.
- Once your robot's motors have been turned on, the robot should go to X, Y -4, 10; then -4, 2; then 0, 2; then 0, -3; then 0, 2; then 4, 2; then 4, 10; then 0, 9; and then repeat.
- After your robot has repeated its X, Y points a few times but a single 2ft by 2ft obstacle in its path. How well does it avoid the obstacle?
- Study the code a bit more to see how this X, Y control code is working. Find the robotdest array in your code. Look in main() for the values its elements are initialized to. The function xy_control() returns 1 (or true) whenever the robot has reached within 0.5 feet of the desired X, Y point. Look at the call to xy_control() in SWI1_HighestPriority() and figure out and **Explain to your Instructor** how the code moves from one point to

another point listed in the robotdest array. NUMWAYPOINTS is currently defined as 8. Change it to 10 and add two more way points (X, Y points) in the path for the robot to traverse. Feel free to add the points anywhere in the array. **Demo this working to your Instructor.**

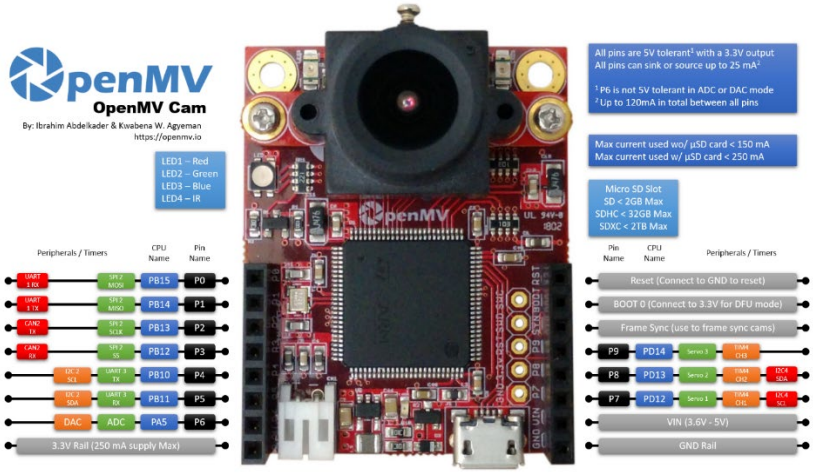
Exercise 2: Up and Running with OpenMV Cam

Hello World

In this exercise, we'll get started working with the OpenMV Cam. The OpenMV project began in 2013, when Ibrahim Abdelkader took it upon himself to build a small, affordable, expandable machine vision module. Unlike the microcontrollers we're used to working with in SE423, the OpenMV uses MicroPython, a lightweight version of Python 3 that includes a small subset of the Python standard library and is optimized to run on microcontrollers. In addition to the standard MicroPython libraries, the OpenMV camera includes extensive machine vision and image processing libraries. To easily interface with the device, debug code, and capture snapshots to find color thresholds, the OpenMV team created an IDE. The IDE is available at OpenMV's downloads page:

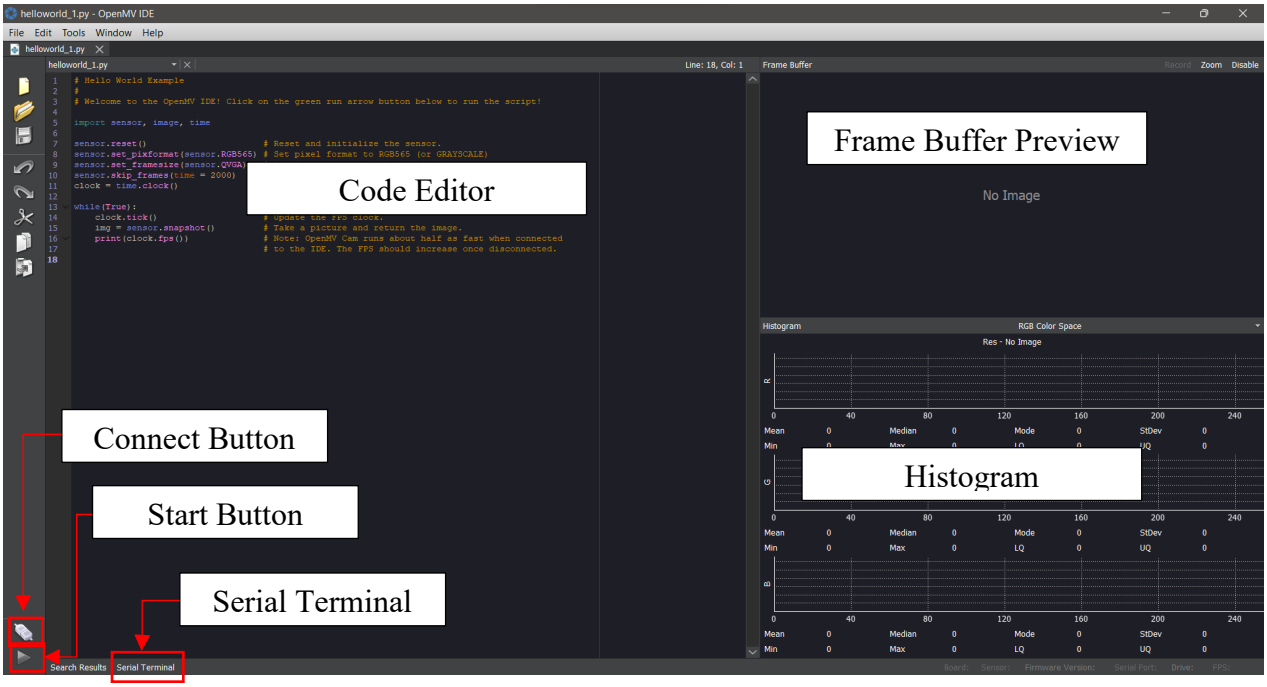
https://openmv.io/pages/download

Specs (If you're curious): The OpenMV H7 Plus camera features a STM32H743VI ARM Cortex M7 processor running at 480 MHz. It incorporates 1MB static RAM and 2MB of flash storage. Its MT9M114 image sensor can record up to 640x480 8-bit images at 40fps, and 320x240 (or lower) images at 80fps. The standard lens 2.1mm, but can be replaced with specialized lenses thanks to an M12 lens mount. Other features of the board include a 12-bit ADC and DAC, 3 I/O pins for servo control, and interrupts and PWM on the board's 10 I/O pins.



First, fire up the IDE on the workstation computers or your laptop. It is ideal if each team has at least one laptop with the IDE installed so that you can work with the camera while it's on the robot on the floor. You will need to capture images of golf balls on the ground to program the color thresholds, and this process is easier if you can directly connect the camera to your laptop. If you don't have access to a laptop, talk to your instructors and we'll show you how to capture snapshots without a direct connection.

When you first open the IDE, you'll see the following window:



The "helloworld.py" script is automatically opened in the editor. Basic file commands such as copy and paste are in the leftmost panel. The "Connect" and "Start" buttons are on the bottom left, as is the "Serial Terminal" button. You'll need to press this to open the serial terminal and view the output of "print()" statements in your code. The frame buffer preview panel is in the top right corner. When code is running on the OpenMV camera, the image currently stored in the module's frame buffer is displayed here. Beneath the frame buffer preview panel is the "Histogram" panel. You can use the dropdown to view the histograms in different color spaces. The IDE also includes some useful information beneath the "Histogram" panel. When a camera is connected, it will display information about the camera, the camera's COM port, the drive that exposes the camera's flash memory or SD card, and the camera's frame rate.

Remove the USB cable connecting the camera to the Raspberry Pi from the camera and connect the camera to the Desktop. You can use the USB cable that we use to connect to the JTAG for the RedBoard. When the camera boots up, it will appear as a USB storage device on the computer. You can use the file explorer to move programs and assets to the camera as well as interact with image saved to the camera's SD card.

Press the connect button in the bottom left of the window (or press Cntrl+E). The "Start" button will turn green, indicating that you can now run the program in the code editor. Also notice that the information in the bottom right of the IDE window is updated. Note the COM port and drive location.

Now, press the "Start" button (Cntrl+R). The "helloworld.py" example runs a while loop that captures an image on each iteration using the sensor.snapshot() method. This command stores an image in the frame buffer, which you should see in the frame buffer preview window.

The frame resolution and pixel format are set using the sensor.set_framesize() and sensor.set_pixformat() methods. Many more camera settings can be adjusted in the sensor library. See the docs here:

https://docs.openmv.io/library/omv.sensor.html

Now, we're going to capture a test image that we'll use to set the color threshold values. Place your robot on the floor with a few orange and purple golf balls in the camera's field of view and connect the camera to the OpenMV IDE running on a laptop. Make sure a golf ball is close to the camera and one is farther away. Again, if you don't have a laptop, ask your

instructor to show you an alternative way to capture the image. Then, load the snapshot.py example file found in File->Examples->OpenMV->Snapshot->Snapshot.py. All this script does is capture an image and save it to the camera's microSD card. Make sure to change the extension of the picture file to .bmp instead of .jpg. Feel free to modify the filename in line 24 or the delay time in line 15. Note that any images with the same filename already on the SD card will be overwritten. When you've made your adjustments to the example file, press the "Connect" button (if you haven't already) then the "Start" button. The camera will delay, turn on the red LED, delay again, then save the picture. In the Serial Terminal panel, you should see a prompt telling you to reset the camera to see the saved image. The camera looks like a normal USB drive to the computer's operating system, and the OS assumes that USB drives can't modify themselves. Because of this, the USB connection must be reset before you can see files generated by the camera in the file explorer. You could reset the camera by ejecting it, unplugging it, and plugging it back in, but your instructor spared you such inconvenience by wiring a pushbutton switch to the camera's reset pin. Press the pushbutton on the front of the camera to reset it. In a few seconds, the OS should recognize the drive. Windows occasionally thinks that there's a problem with the drive; you can ignore its prompts to scan and fix the drive. Open it up in Windows file explorer and copy your image to the computer. Before we go to the next step of finding a good LAB threshold value for these two golf ball colors it is nice to first find an approximate LAB range for each color. To find a LAB starting range, reload the "helloworld_1.py" OpenMV program again and run the code. While OpenMV is displaying your image with the golf balls present on the floor, with your mouse click and drag a small box that just includes pixels with the color you want a LAB threshold range. Once you do this the histogram displayed on the right is just displaying the LAB range of the pixels inside that box. Look at the Min and Max values of the histogram and that will tell you a starting range for your LAB threshold. Do this for both colors and write down this initial LAB threshold before going on to the next paragraph.

In the IDE, navigate to Tools->Machine Vision->Threshold Editor and load the image file you just saved. This will open a window containing your source image on the left and a binary image on the right. In the binary image, white pixels are in the threshold and black pixels are not in the threshold. That is, you want your golf ball of interest to be white and everything else to be black. Use the min/max sliders for the L, A, and B channels to adjust the threshold. Recall from lecture that the "L" channel is lightness, the "A" channel ranges from green to magenta, and the "B" channel ranges from blue to yellow. Ask your instructor for some tips on finding the correct LAB ranges. When you are satisfied with the threshold, copy or write down the threshold values towards the bottom of the window. We will use these values in the blob search code so the camera knows which colors to look for. Do this for both color golf balls.

Exercise 3: Manual Blob Search

Next, we'll create a crude blob search program that scans through each pixel in the image and recolors all pixels, within the threshold, to black. It also calculates the centroid of the detected pixels and draws crosshairs at that location.

In your repository there is a folder called "OpenMV". For this exercise you will load from that folder the file ManualBlob_STARTER.py. In lines 7-14, assign your threshold values to the L, A, and B min and max values. Recall that the order of the six threshold values is: (L_min, L_max, A_min, A_max, B_min, B_max). Scroll down to the main loop at line 30 and notice the same sensor.snapshot() method used to store an image in the frame buffer. This method returns an "image" object that we can use to interact with the image. Next, you will need to create "worker variables" that you will use to calculate the centroid of all pixels within the threshold. Review your notes from lecture on what worker variables are needed.

Next in the code is the two nested for loops that iterate over the columns in the outer loop and rows in the inner loop. Within these loops, the image.get_pixel() method is used to get the RBG value of the pixel at column=i, row=j. Next, the rgb_to_lab() method is used to convert the RBG values to LAB values. In line 49, we check to see if the pixel falls within the threshold. If the pixel does fall inside the threshold, you will need to add this pixel to the sum of pixels already found within in the threshold. Your worker variables will be used to keep track of these pixels. Note: one of your worker variables will need to be called "pixel_cnt".
After the for loops, use your worker variables to calculate the centroid of the image. X_cent = ? and Y_cent = ?

Once you've made the required changes, connect to the camera and run the program. You should see an image in the frame buffer preview along with the centroid and frames per second printed to the serial terminal. Verify that the correct pixels were detected, that is, the region of interest is black, and notice the white crosshairs in the center of the detected region. Also notice that this program runs very slowly. Looping over an image like this is very inefficient, especially when implemented in a relatively slow language like Python. In the next exercise, we'll see how OpenMV's built-in blob search algorithm can get the job done much faster. One quick additional exercise: instead of painting the pixels within the threshold black, paint these pixels green instead.

MultiBlob

For this exercise you will use/open the other python (.py) file in your repositories OpenMV folder MultiBlobwithLCD.py. This script uses the image.find_blobs method to find the three largest blobs that fall within a color threshold. In addition to highlighting the blobs and drawing crosshairs at the centroids, it sends the areas and centroids over UART to the RedBoard. For now, we can ignore the communication protocol while we modify our vision algorithm. First, enter one of your color threshold values into the "threshold" variable in line 13. Connect the camera again and run the script. You should see a rectangle drawn around the desired object with a cross hair in the center. Notice how much more quickly the built-in blob search algorithm performs and notice the frames per second (fps) printed to the terminal. We have a few additional knobs here to adjust the blob search algorithm. The "pixels_threshold" and "area_threshold" arguments of the "find_blobs" method set a minimum allowable area for the number of pixels in the blob and the area of the blob, respectively. If the algorithm doesn't find the object, try reducing these threshold values until it reliably finds the object. If the algorithm picks up extraneous blobs smaller than the desired object, try increasing the values. If none of these methods work, you may need to go back and verify your threshold values. Play with these two thresholds so you see how they work. **Demonstrate this code working to your TA.**

Exercise 4: Interfacing with F28379D Board and Distance Calculation (This section will be finished Week Two)

In the previous exercises you used the OpenMV's IDE to debug and download micro-Python programs to the OpenMV board. We will again use this IDE to modify starter code to search for both orange and light purple golf balls. After we have our OpenMV code ready to run continuously, the below steps will explain how to save the Python code on the OpenMV's SD card so that when it is either powered on or reset, with a press of the reset button, it will be the default code running.

1. The OpenMV code we will use for Ex. 4 and Ex.5 is again in your repositories OpenMV folder and called MultiBlobtwoColorwithLCD.py. Open this file in the OpenMV IDE. Again it will be the best if you can use your own laptop to run the OpenMV IDE so you can move the robot to the floor. Either copy the MultiBlobtwoColorwithLCD.py file to your laptop or pull the latest code in your repository. Looking through the

given code, it performs the "find_blobs" algorithm to search for two color thresholds. It then sends the largest three blobs of threshold1 and the largest three blobs of threshold2 to the F28379D board over UART. If it does not find all three blobs of a color, it sends Area = 0, Column = 0, Row = 0 to indicate that this blob was not found. Also notice that the given thresholds are not orange and purple. As a first step change "threshold1" so that it is the color range you found for the orange golf ball in Exercise 3. Change "threshold2" to the range you found for the purple golf ball. Run this code from the OpenMV IDE and put the robot on the floor in the course. Put both orange and purple balls in the field of view of the camera and make sure on the LCD screen that both colors are found and cross hairs are placed on the blobs. Move the robot around the course and make sure nothing else on the floor besides golf balls are recognized by the camera.

2. Now that the OpenMV code seems to be working well, we would like to make this code the default code that the camera runs after power on or reset. To do this all you have to do is copy this MultiBlobtwoColorwithLCD.py to the OpenMV's SD card and rename the file "main.py". When you connect the OpenMV camera to your laptop's USB port, it should bring up a folder. This folder is OpenMV's SD card. Look at the files and notice that one of the files is "main.py." So first copy your MultiBlobtwoColorwithLCD.py file to the OpenMV SD card. Then in the SD card delete the main.py file. Finally rename MultiBlobtwoColorwithLCD.py to main.py. Once saved as main.py you can either press the OpenMV's reset button or unplug the OpenMV camera from USB and plug it back in. Once you do this your main.py code will start running. Notice how the OpenMV's blue LED is just on for a short burst about 2 times a second.
3. Before you go on and write code for the F28379D to receive this centroid data from the OpenMV camera, notice one more thing about the OpenMV camera. The OpenMV does not need to use a SD card. Remove the SD card from the camera and power on the OpenMV without the SD card. Notice that it also pulls open a folder. Also notice that the blue LED is blinking at a different rate. Technically you could copy your code to the main.py file directly on the OpenMV and it would run the same as if you copied it to the SD card. When first working with the OpenMV we found ourselves copying code that we thought was bug free to main.py but then found out later that we had some errors that caused the OpenMV to crash. For that reason I want you to leave the main.py file that is on the OpenMV itself alone and only change the main.py file that is on the SD card. When the SD card is inserted it is the only main.py file that the OpenMV uses. But if you make a mistake in that main.py file on the SD card you can power off the OpenMV camera, remove the SD card and power back on the OpenMV camera to see that the default main.py code still runs. This makes for a good "sanity" check to see that the OpenMV is still working.

- Now let's move to code on the F28379D board. The starter code is all set to read the centroid data sent from the OpenMV camera to the F28379D's UART_SCIA. All you need to do is flash this starter code to the F28379D. Also your TA will show you how to plug the OpenMV camera's USB port into one of the Raspberry PI's USB ports. With the code flashed to the F28379D, place the robot on the floor in the course and press the reset button of the F28379D along with the reset button of the OpenMV. Once running put some golf balls in the camera's view and you can see the "Max Blob Area, Max Blob Column Centroid, Max Blob Row Centriod" printed to the text LCD screen if you press button 1. The top line is the orange color and the second line is purple. Move the robot around a bit and notice the values changing on the text LCD. Put three purple and three orange balls in front of the robot and now press button 2. Notice the second largest blob coordinates are printed to the text LCD. Finally if you press button 3 the third largest blob coordinates are printed to the text LCD. **Show this working to your instructor.**

- Now that you have areas and centroids printing to the text LCD (when you press the correct button), collect some data to find an equation that relates row centroid to the distance the golf ball is away from the robot. One very important assumption you will be making here is that the golf ball is always on the floor. Also since the camera has a fish eye lens, you will make sure the ball is in the center columns of the camera image. Ask your instructor for a yard stick and use it to measure the distance the center of the robot is away from the golf ball. Take measurements every 0.5 feet, starting at 1 foot, up to 4.5 feet away from the center of the robot. Record the distance from the robot and the row centroid at that distance. Then when you have the data recorded, use curve fitting in Matlab to find a good equation that calculates the distance in feet from the robot to the ball given the ball's row centroid. This will not be a linear equation. Calculate this equation in your SWI1 function when a new centroid is found and every 100ms print the distance the ball is away from the robot to the text LCD screen. **Demonstrate this working to your instructor.**

Exercise 5: Color Following (This section will be finished Week Two)

- For this exercise you will start to implement RobotState 20 (Follow/Pickup Orange Golf ball) and RobotState 30 (Follow/Pickup Purple Golf ball). Looking at SWI1's code you will notice there is an empty RobotState 20 case statement. You will be adding a case 30 later. For now let's just focus on following the largest orange blob the OpenMV finds. For this reason at the line before the statement "switch (RobotState) {", add "RobotState = 20". This will make the state machine code always run state 20. Inside state 20 keep the code pretty simple for now. The goal is to follow the orange golf ball so it always stays in the center of the camera's view. Because we are interested in the center of the camera's view left to right, the following code will use the column centroid. Also since the goal is to keep the blob in the center columns of the camera's view, it is nice to change the centroid so that zero is in the center of the view. So create a global float variable colcentroid and since our camera image has 320 columns set it to colcentroid = MaxColThreshold1 – 160; colcentroid will now be a value between -160 and 160. Additionally inside case 20 implement the following code listed as Pseudocode:

```
If MaxColThreshold1 == 0 || MaxArea Threshold1 < 3
        Vref = 0
        Turn = 0
Else
        Vref = 0.75
        Turn = kpvision * (0 – colcentroid)
        // start kpvision out as 0.05 and kpvision could need to be negative.
```

  Run this code on the floor pushing an orange golf ball around making the robot follow. Tune your kpvision value to make the robot flow nicely.
- With the robot being able to follow orange, modify RobotState 1 and RobotState 20 to do the following:
  a. Get rid of the hard coding of RobotState = 20 at the top of the state machine code.
  b. In RobotState 1, if MaxAreaTheshold1 is great than some area, switch to RobotState 20. To find this area, put the robot on the floor and move a golf ball towards the robot to the distance when you would want to start following towards the golf ball. Use this as the area to switch to RobotState 20.
  c. In RobotState 20: here use the code you developed above that has the robot turn towards the golf ball and drive towards it keeping the largest orange blob in the center columns of the camera's view. If

MaxRowThreshold1 > "a large (bottom) row number" (you will need to play with this number) then switch to RobotState 22.

d. In RobotState 22: set Vref and turn to zero to stop the robot. This is simulating you waiting for your gripper door to open. Keep track with a count variable how long you have been in RobotState 22. When 1 second has gone by switch to RobotState 24.
e. In RobotState 24: set turn to zero and Vref to 0.5. Keep track with a count variable how long you have been in RobotState 24. When 1 second has gone by switch to RobotState 26.
f. In RobotState 26: set turn and Vref to 0. Keep track with a count variable how long you have been in RobotState 26. When 1 second has gone by switch back to RobotState 1 and the robot will continue to its X, Y point.
g. Using a similar count variable, modify RobotState 1 so that once you switch back to RobotState 1, you cannot switch to RobotState 20 or 30 (coming up in the next bullet) for 2 seconds.
h. Finally make sure "RobotState", which is an int16_t, is printed to the text LCD.
i. **Show this working to your instructor**

- Create a case 30, 32, 34, 36 that performs all the same steps as 20, 22, 24,26 but for purple and in case 1 switch to case 30 if the largest ball found was a purple golf ball. You will also have to change your case 1 switch to case 20 to check both orange and purple to see which is the largest if both are seen. **Show this working to your instructor.**

Lab Check Off: Show your TA the final F28379D program and LABVIEW program commanding the robot to go to the different x, y points and then if a golf ball is seen go towards the golf ball until it disappears. Stall for two seconds once it disappears and then return to commanding the robot to x, y points.

How-To Document: Create a document that reminds you of all the steps you need to remember to run the OptiTrack software Motive and its additional scripts, working with the OpenMV camera and interfacing the OpenMV camera with the F28379D board.