CymSTAR Capstone Joystick Controller

Compilation of Resources for the Software User Manual (SUM), Software Product Specification (SPS), and Software Design Document (SDD)

Clark Shannon, Connor Redington, Gracen Ownby,

Matthew Crawford, and Michael Fergusen

Table of Contents

Downloading Arduino Software and Uploading Firmware to Device

To recognize and connect to the Arduino, download the Arduino IDE software onto the Host PC and run it to install the drivers needed to connect to the Arduino. The most up-to-date version depending on the Host PC's operating system can be found here:

https://www.arduino.cc/en/software

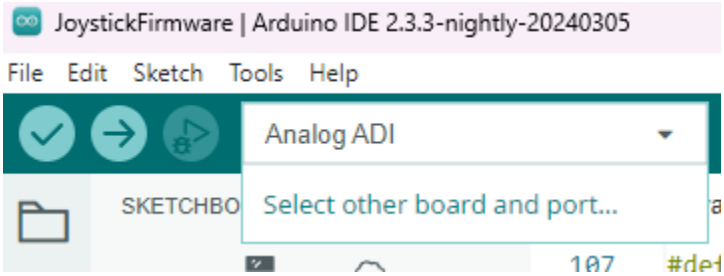
This IDE is used to develop firmware to then be uploaded as a "Sketch" to the Arduino device. Our firmware code can be found on the Redmine repository in the directory below:

joy_con_repo > Arduino > JoystickFirmware.ino.

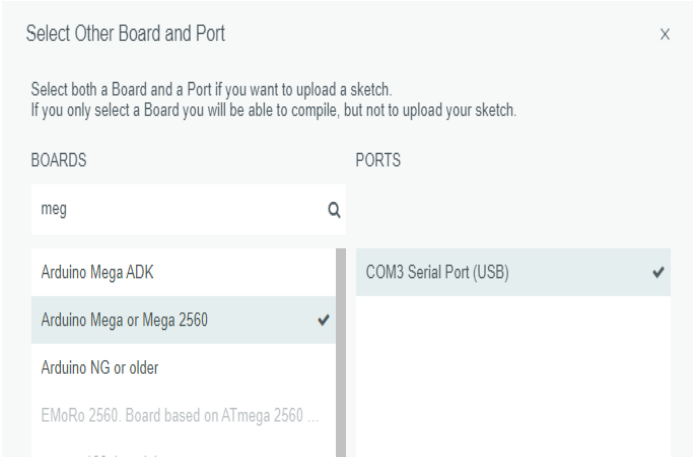
To open this code in the Arduino IDE, you need to open up the Arduino IDE application and navigate to:

File > Open >…>joystick-controller.joy_con_repo > ArduinoFirmware > .ino file

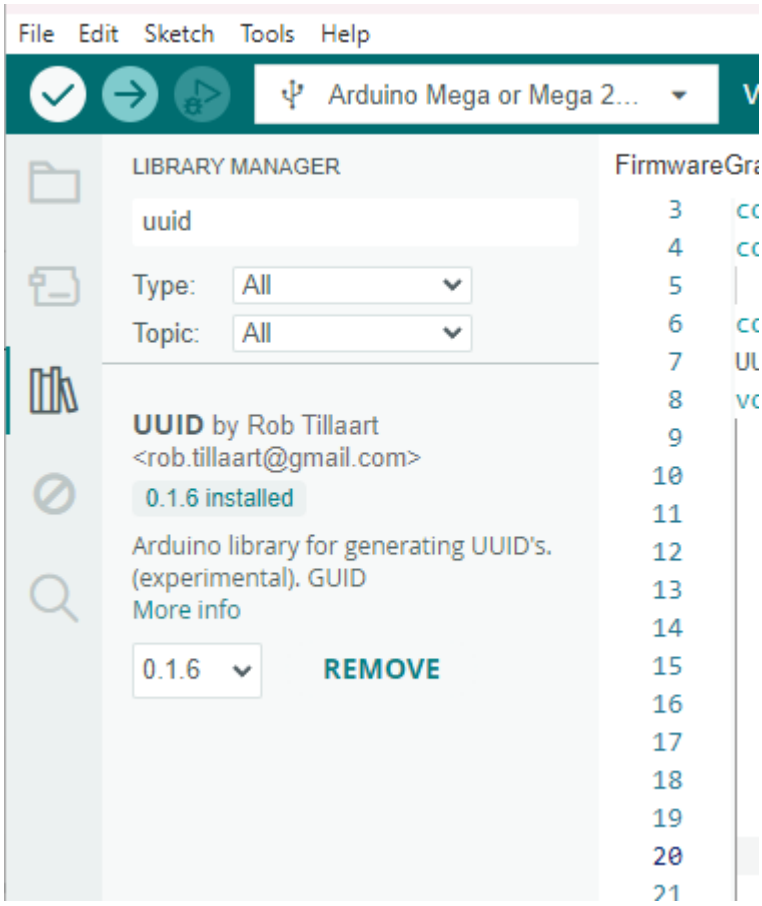
This will open the file in a new window. At the top of this window, you can verify, compile, and upload the code to the device by first clicking the dropdown menu in the top lefthand corner:



Clicking the "Select Other Board or Port" option will bring you to the menu below, where you can search and select the Arduino Mega 1560 as well as the associated port that should automatically populate when the board is plugged in.



Once this is done correctly, you will additionally need to add in the UUID library (Reference: UUID). This is done by going to the book icon in the left panel, searching “UUID”, and then installing the library. Once this is complete, you should see it showing up as installed in the library manager:



After this is installed, the code is ready to be compiled and uploaded to the device. First select the “Verify” checkmark in the top left corner shown in the picture above. Once this is completed, the code can be uploaded to the Arduino Mega device by clicking the Upload button (Arrow pointing to the right). A notification will pop up in the bottom right of the window saying “Done Uploading” when this is complete. It is important to not have the serial monitor or the python application running while trying to upload as it will not allow the upload to occur while the serial communications are being processed over the USB. This IDE will be the place where any changes to the firmware are made but unless any are needed, there will not be anything else to do for the code to run on the device besides having the USB cord connected to the Host PC.

Setting up the Python Environment

After the firmware has been properly uploaded to the device, the main processing of the solution will take place in a Python Environment to run two python files. These two files can be found in the Redmine Repository for our project by going to

joy_con_repo > HostComputer > code

The files in this folder need to be opened within a code editor such as Visual Studio code or PyCharm. The following need to be installed before being able to run this program. The following example will show the complete steps needed for someone who has not ran a python program before:

1. Download and install a Code-Editor such as Visual Studio code (Download)
2. Install a Python interpreter: https://www.python.org/downloads/
    a. Check to make sure this is done properly by opening either a command prompt or Terminal window:
        i. Terminal (Linux or macOS): python3 --version
        ii. Command Prompt (Windows): py -3 --version
3. Open up the file folder above named code in this Code-Editor
4. In the terminal in this Python environment, enter the following commands to install the needed libraries:
    a. pip install -r requirements.txt
5. Once this is done, the Python environment should be established to now go and make any changes necessary to the .py files

The GUI itself is initialized within the JoystickGUI.py file, whereas the serial instance created and established with the Arduino is determined in this file by calling a function defined in the ArduinoConnect.py file. To change things like baud rate or other configurations of the serial instance, these need to be made within the ArduinoConnect file and then replicated in the setup function of the firmware of the Arduino. Any changes to the number of I/O, how they are displayed, or the order in which they are passed from the Arduino will also need to be reflected in the Python file. Each data point is printed sequentially on the Arduino and then read in the same order in JoystickGUI, meaning any change in the processing of data on one set of code will need to be reflected in the other.

Overview of Complete Software Solution

Our software is comprised of two sets of code: a firmware .ino file compiled and uploaded to the Arduino Mega device and an executable built from two python scripts. The firmware code will execute repeatedly after the device is connected to the Host PC using the USB cable. The python executable will, once started, begin communicating with the Arduino device via Serial communication configured using the same properties on both systems. Once serial communication is established, the user's input on the hardware device is read and transmitted to the host computer to be displayed in a Graphical User Interface (GUI) built by the python program. This GUI runs continuously while the Arduino is connected to the Host PC and displays the real-time status of the inputs read by the device and accepts input from the user to control the digital outputs connected to the Arduino. The python program reads in the data from the Arduino line by line, decoding each line sent from the Arduino in the same order it was transmitted and displaying it in the corresponding widget of the GUI. The GUI will continue to display the real time data from the Arduino until the device is disconnected or the GUI window is closed.

The execution of these two sets of code can be broken into three different stages:

1. Setup and Initialization:
    a. Firmware: input and output pins connected to the board are defined and serial communications are opened
        i. # define digitalInput28 49; Defines a variable named digitalInput28 to be connected to pin 49 on the device.
        ii. pinMode(digitalInput28, INPUT); Designates this pin to be an input in the system. Reference: pinMode()
        iii. Serial.begin(9600): opens up serial communications with a baud rate of 9600 bytes per second and then waits until a connection is made (while (!Serial){}). Serial.begin()
    b. Python GUI: The window and all widgets on it are created, configured, and organized to display the inputs and outputs connected to the Arduino.
        i. This is done using the python package tkinter, which is a platform independent windowing toolkit. Reference: tkinter:
        ii. A tkinter window is created and widgets are added to this window.
            1. window = tk.Tk(): Window is created and contains all the elements added to the GUI. It is the root object on which all other elements are created and placed. Intro
        iii. The widgets used in the GUI are frames, buttons, labels, and scales
            1. Frame: frameA = tk.Frame(parent): a widget used to organize the user interface and upon which other widgets are added. Frame
            2. Button: buttonB1I = tk.Label(root, text="Digital Input 1"): used to accept user input when a person clicks the button. Command is

attached to the button to execute whenever the button is pressed. Button

3. Label: label = tk.Label(frame, text="Digital Ouput"): labels are text objects used to display the current values of the analog inputs and to act as virtual leds, flashing red and changing to "ON" when a button is pressed. Label
4. Scale: sliderS1 = tk.Scale(frame, from_=0, to=5): slider is a visual widget used for our potentiometers, moving the scale from 0 to 5 depending on the voltage reading from the analog inputs. Scale

2. Establishing Communications
    a. Firmware defines a device ID. It opens the Serial port and begins sending this device ID out over the serial bus and listens for a response. If the device is recognized by the Host PC, the firmware reads the response and begins the data transfer loop.
        i. Until the confirmation from the Host PC has been received, the Arduino will continue to print the device ID on the serial bus, listen for an input from the HostPC, and then check this input against the confirmation flag.
            1. Serial.println(deviceID): Used to transmit the device ID over serial to the python program
            2. if (Serial.available() > 0): Checks to see if there is incoming data on the serial bus to be read. Serial.available()
            3. userInput = Serial.read(): Reads in the data in the serial buffer and stores it in the variable userInput. Serial.read()
            4. If that input is the flag 'a', a Boolean value will be set to true and the data transfer loop will begin
    b. Python GUI Program: A function named findArduinos() defined in a python script named ArduinoConnect.py is called at the beginning of the JoystickGUI.py program and establishes the connection to the Arduino. This is done using the python package pySerial: pySerial, pySerial functions
        i. It determines all the ports connected to the Host PC and checks the drivers' names to see if they contain the word 'Arduino'.
            1. ports = serial.tools.list_ports.comports(): This is a built in function in from pySerial that returns a list of COM ports connected on the HostPC. list_ports
        ii. If an Arduino is connected, a serial instance is created and the function reads and decodes the first line being sent from the Arduino.
            1. serialInst = serial.Serial(commPort, baudrate=9600, timeout = 1): Serial instance is created using the pySerial package with a baudrate matching that defined in the firmware and a timeout limit of 1 s.
            2. idVal = serialInst.readline().decode('ascii').strip(): Line is read from the serial instance, decoded and stripped of whitespace to match what was sent in the firmware

iii. If the proper unique device ID is read, the connection is established and the confirmation character ‘a’ is sent to the device to indicate it is ready to accept data.
    1. serialInst.write(b'a'): Writes the character ‘a’ to the serial connection to indicate to the firmware to begin sending data.
    2. serialInst.write(b'x'): Writes the character ‘x’ to the serial connection to indicate to the firmware end the serial connection and go back to the setup function.
iv. A Tkinter Window containing the GUI and all Joysticks, Inputs, and Outputs is displayed..
    1. serialConnections = ArduinoConnect.findArduinos(): The line calling the findArduinos() stores the serial instance object created for each Arduino connection and is used to communicate with the Arduino throughout the Python GUI program.

3. Data transfer and displaying
    a. Firmware: Once the connection has been established, the firmware’s infinite loop begins to execute, each iteration following the same process:
        i. Check if any data is being sent from the Host PC. If there is, read in this data and store in a variable called userInput.
            1. if(Serial.available()>0): Checks to see if there are any incoming data bytes in the buffer to read. Serial.available()
            2. Serial.read(): Reads the data from the buffer and returns the ASCII decoded data. Serial.read()
        ii. Read in and store the value for each digital and analog input connected to the device.
            1. analogRead(analogInput): analogRead()
            2. digitalRead(digitalInput): digitalRead()
        iii. Write out to the digital outputs (LEDs) to turn them on or off based on input from user on the GUI that was read in.
            1. digitalWrite(digitalOutput1, HIGH); Sets the voltage at the digital output 1 pin high, turning on the LED. digitalWrite()
        iv. Print the data for all the inputs to the serial connection line by line, starting with a start delimiter character ‘<’ and ending with a stop delimiter ‘>’ to ensure data order is preserved.
            1. Serial.println(): Serial.println()
    b. JoystickGUI: After the connection is established and window has been initialized, the program begins a loop to run concurrently with the firmware loop and follows a similar process, sending data to the Arduino at nonregular rates only when a button on the GUI is pressed by the user.
        i. Read in data packet from Arduino and check to make sure data is valid. If it is, store the data in variables to be used to update the GUI. If not, pass over the loop and wait for the next valid packet of data

1. This is done in a user-defined function called requestData(): It is called at the beginning of each loop and starts by reading in each line from the serial buffer and checks for the start delimiter '<'. If the start delimiter is not read, it does nothing and reads the next line in the buffer. Once '<' is read, the code begins storing each line read on the serial buffer in a python list. Once the stop delimiter '>' is read, the program stops reading in data and returns the list to the main loop.
    a. Python lists: lists
    b. Serial readline(): readline()
2. The list returned from requestData() is checked to make sure it contains all 40 data points sent by the Arduino; if it does, it is assigned to variables to display the corresponding inputs in the GUI
    a. The first 8 elements of this list are converted to floats and stores in their corresponding variables:
        i. serialDataX1 = float(inputData[0]) # Joystick 1 X-axis
        ii. serialDataY1 = float(inputData[1]) # Joystick 1 Y-axis
    b. The next 32 are digital inputs from the pushbuttons. For each value that is high, the label created representing a virtual LED for that button on the GUI is turned "ON!":
        i. Elements 8 through 39 from the input list are checked, calling the function turnOnButton for each that is logic high meaning that the button was pressed
        ii. label.config(text="ON!", bg='red') # Turns label "ON!", changing the background color to red

ii. Determine the coordinates to plot for each of the joysticks and which buttons were pressed based on this data.
    1. The x and y coordinates for each joystick are read in from the Arduino as voltages ranging from 0 to 5 V. To plot them on a four quadrant graph, we convert the voltage to a range from -1 to 1 by subtracting or adding 2.5 to the original value and then multiplying by .4
    2. The plotting is done using the matplotlib.figure package and the matplotlib.backends.backend_tkagg package.
        a. matplotlib.figure: this package allows us to create a figure with customized settings to plot the x and y position of each joystick.
            i. joystickFigure1 = matplotlib.figure.Figure(): Creates the figure to hold the position plot. matplotlib.figure
        b. matplotlib.backends.backend_tkagg: This package allows us to integrate the matplotlib figure into the tkinter window we are creating to be displayed alongside the Tkinter widgets we use for everything else in the GUI. The matplotlib figure is

attached to a Tkinter canvas allowing for it to be displayed like any other widget in the GUI

i. joystickPlot1.scatter(x,y): This is a matplotlib function used to create the scatter plot of the x and y coordinates read from the joystick. Scatter plot

ii. canvasPlot1.draw(): This updates the canvas the figure is attached to and updates the display in the GUI. Canvas Tkinter

iii. Update all GUI widgets based on most current values and update the window itself.

1. Update the Scale widgets based on the reading from each potentiometer.
    a. sliderS1.set(serialDataPot1): Sets the position of the scale object. scale.set
    b. The label for each scale is also updated with the current reading:
        i. updateLabelVal(labelS1, "Potentiometer 1: %.2f V" % serialDataPot1): this changes the text configuration of the label object labelS1to include the variable serialDataPot1, formatted to be limited to 2 decimal places. label.config
2. Update the labels for each matplotlib figure: Same process as above with the potentiometer labels.
    a. updateLabelVal(plot1Label, "Joystick 1: X Position: %.2f; Y Position: %.2f" % (positionX1, positionY1))
3. The buttons on the GUI window have functions attached to them to execute whenever the button is pressed. Each button has an associated character that is written to the serial connection when that button is pressed.
    a. def turnOnLED11():
        serialConnection.write(b"w"): This is decoded in the Arduino firmware to know to turn on the associated LED. pyserial write

iv. Reset the plots:

1. joystickFigure1.clear(): clears the plots to be able to plot the updated position in the next iteration of the loop. matplotlib.figure.Figure.clear()