CameraX Arduino-based Controller Design & Assembly

C. Portmann, L. Shaw

06/06/2019
Doc revision: 1.0

Table of contents

Overview

The CameraX test lab requirements enable a cheaper, easier-to-manufacture controller. The “sensor fusion” test system, which was used for the initial CameraX lab build, provides a proscribed motion of the phone for camera/gyroscope timestamp testing. The speed of the motion is critical, and the motion must be smooth as images are captured during movement and phone position is inferred from the images to compare with gyroscope events. Design details are located here: AOSP sensor fusion box details.

For CameraX, however, the speed and smoothness of the rotation is not important, but rather the ability to stop at various positions to take ‘landscape’ and ‘portrait’ images with the device under test. With a CameraX feature-set focus, an Arduino UNO-based controller was designed. The Arduino-based controller provides motor control for the existing sensor_fusion test rigs, but requires new control software and custom micro-code for the Arduino. Additionally, an Arduino UNO can control up to six separate motors rather than four like the sensor fusion controller, improving the host-to-test-rig ratio.

Figure 1: Arduino UNO R3

Arduino UNOs and motor control

Figure 1 shows an image of the Arduino UNO R3. This is one of the most popular boards from Arduino and is externally controllable through a USB-B connection. Additionally, the drivers to control the UNO through USB are available by default in Goobuntu, eliminating any customization required by the hosts. Smaller, cheaper Arduinos, like the Nano, require custom drivers installed on the host and add to the complexity of the project.

Analog servo motors are controlled using pulse-width-modulation (PWM), and the UNO has 6 PWM outputs that are individually customizable. The PWM outputs for the UNO are DIGITAL pins shown with a '~' on the right side of the image: pins ~3, ~5, ~6, ~9, ~10, and ~11. Finally, the Arduino programming environment comes with a built-in servo library that allows one to program the angle of the servo.

The motor used is the HiTEC HS-755HB shown in figure 2 along with the PWM control diagram. The HS-755HB motor is an analog quarter-scale positional servo with Karbonite™ gears to improve reliability. Control is applied through a 3-pin connector with SIG/VCC/GND. Cables are color coded. Yellow is SIG, red is VCC, and black is GND. VCC is 5 Volts. SIG is the PWM signal described in figure 2. The HS-755HB spec sheet has a throw of approximately 200 degrees with pulse widths varying from 500 to 2500us. Note the rotation appears to be the opposite direction of the sensor_fusion controller.

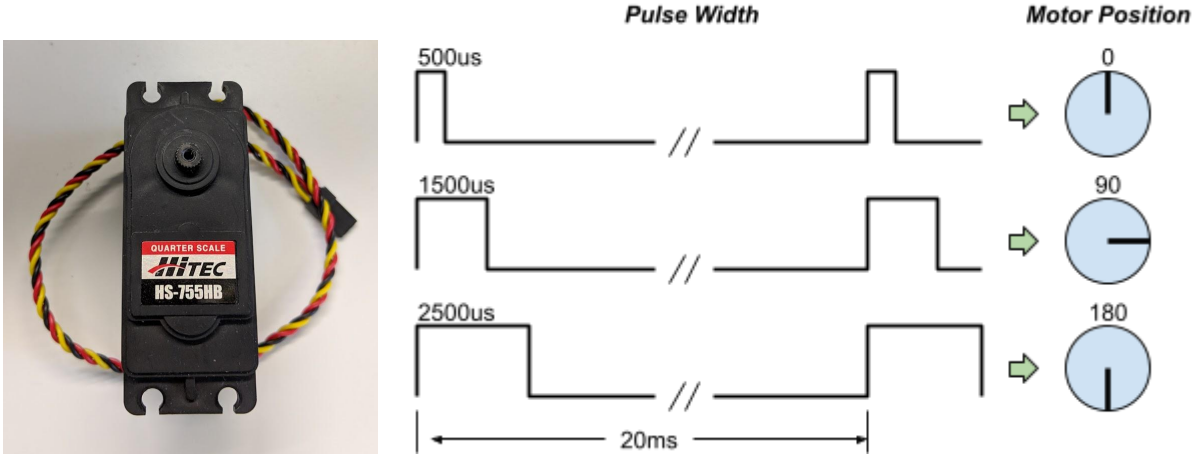


Figure 2: HiTEC HS-755HB servo motor and PWM diagram.

Arduino routing shield

A two-layer routing board has been designed to mount on top of the Arduino UNO controller and interface with six servo motors as shown in figure 3. Gerber file download link. The top view shows the motor headers in the center along with the lighting control connector at the top. (Lighting control will be described later in this document.) The bottom view shows the header connections required to mate with the UNO, the 5-V power jack and a 10-uF bypass capacitor. To isolate the motor currents from the Arduino power supply, VCC for the motors is provided through the external 5-V jack. Motors starting and stopping can create large transient currents on the supply. The UNO electronics are powered separately through the USB connector and there is no sharing of VCC between the two boards, only GND. The 5-V power jack on the UNO is unused and in the enclosure is covered up to avoid confusion.

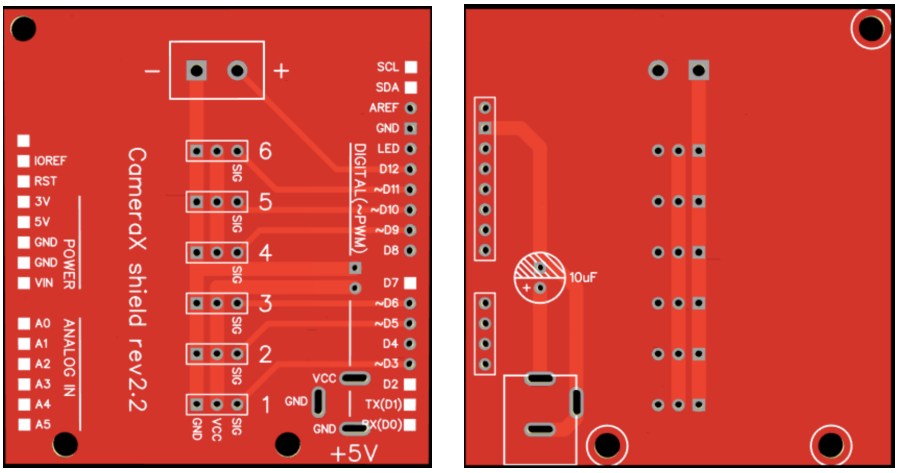


Figure 3: Top and Bottom views of CameraX rev2.2 Arduino shield.

Enclosure

A custom enclosure was designed for the controller. The mechanical drawing is shown in figure 4. The assembled controller mounts to the bottom of the enclosure through four counter-sunk screws through the bottom plate of the encloser. Pertinent information (motor signal numbers, lighting control polarity and 5V inpuT) are etched into the plastic top.

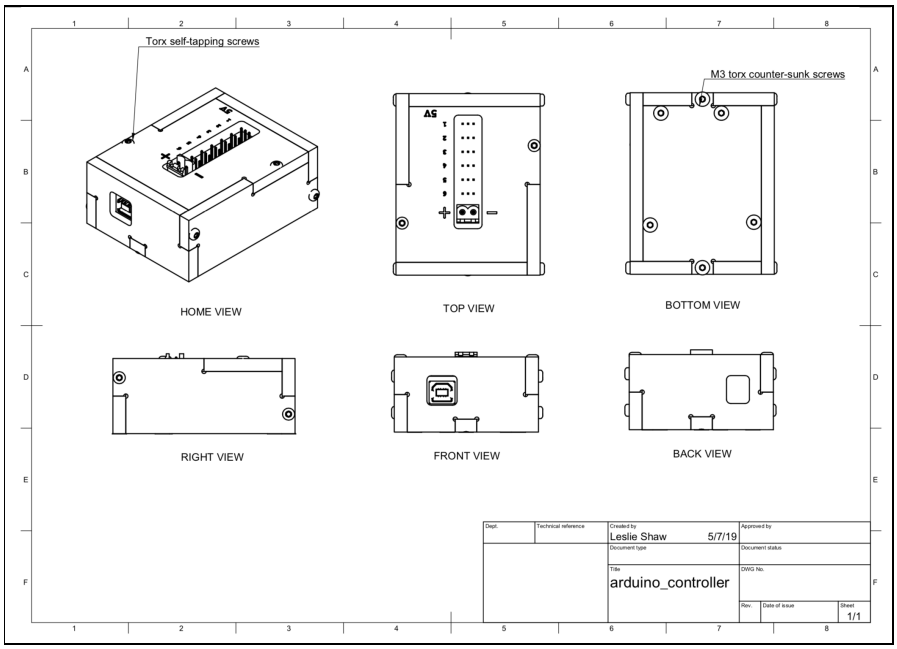


Figure 4: Enclosure design

BOM

Qty	Item	PN/Link

1	Arduino UNO rev3	https://store.arduino.cc/usa/arduino-uno-rev3
1	Google CameraX routing board (1.6 mm thick)	
1	Google CameraX controller enclosure	
1	2.1x5.5 mm, 5 V through-hole, right-angle barrel jack	101179
1	10 uF, 16 V, 10% tantalum capacitor	TAP106K016CRS
1	1x2x, 200 mil (5.08 mm) pitch, through-hole Phoenix Contact header	651-1755736
1	1x2x, 200 mil (5.08 mm) pitch, pluggable Phoenix Contact connector	651-1757019
6	1x3x, 100 mil (2.54 mm) pitch, through-hole male header	952-3308-ND
1	1x8x, 100 mil (2.54 mm) pitch, through-hole male header	952-3308-ND
1	1x4x, 100 mil (2.54 mm) pitch, through-hole male header	952-3308-ND
4	M3-0.5, 6 mm male-female standoffs (5 mm width)	R30-3000602
3	M3-0.5, 11 mm female-female standoffs (5 mm width)	R30-1001102
3	M3-0.5, 6 mm screws	36-9191-3-ND
4	M3-0.5, 8 mm flat-head screws	mcmaster
1	5 V, 15 W UL-listed power supply, 2.1x5.5 mm plug (motors)	

Other tools needed

- Soldering iron and solder
- Small Phillips-head screwdriver
- Size T10 Torx screwdriver

Controller assembly

Populate the top and bottom of the routing board with the parts fitting in their outlines. For the bottom of the board, the male headers can be aligned by placing the headers into the correct locations in the Arduino board and placing the routing board atop the connectors. The 1x8 and 1x4 headers can then be soldered in place, guaranteeing good alignment between the Arduino and the routing board. The same can be done for the power jack, but a shim will be needed for tight assembly. After soldering the bypass capacitor, the top of the board can be populated with

the six, 1x3 male headers for motor control and the 1x2 Phoenix Contact header for lighting control.

Once all connectors and components are soldered in place, the system can be assembled using the standoffs and screws. Note, there are four male-female 6mm standoffs to provide connection and mechanical stability to the bottom of the plastic enclosure. However, there are only three female-female standoffs due to one hole on the Arduino, the one near the SCL pin, being unusable due to proximity of the hole to the female header. Screw the three female-female standoffs on to three male-female standoffs to secure them to the Arduino. Then screw the routing board shield on to the standoffs with the three, M3 screws. The four male-female standoff can be screwed to the mounting plate to give mechanical support to the fourth corner of the Arduino. Figure 5 shows a conceptualized end view of the populated CameraX board. Figure 6 shows the a populated and mounted system. Note the Phoenix connectors should be oriented so that the bottom of the snap in connector is towards the motors to give maximum room for motor mount. Figure 7 shows the controller inside the enclosure.

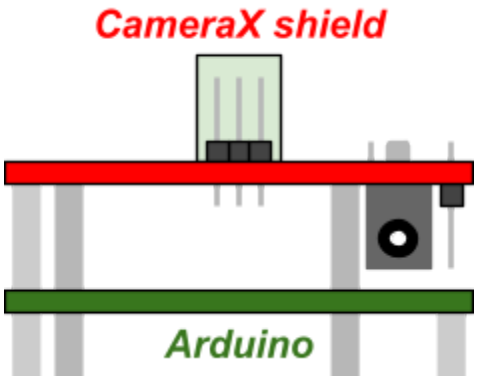


Figure 5: Conceptualized end view of populated CameraX board on Arduino showing stacking.

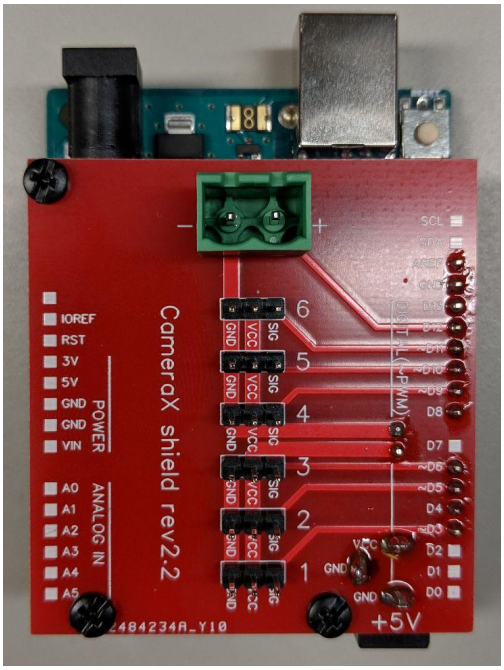


Figure 6: Populated and mounted rev 2.2 Arduino shield.

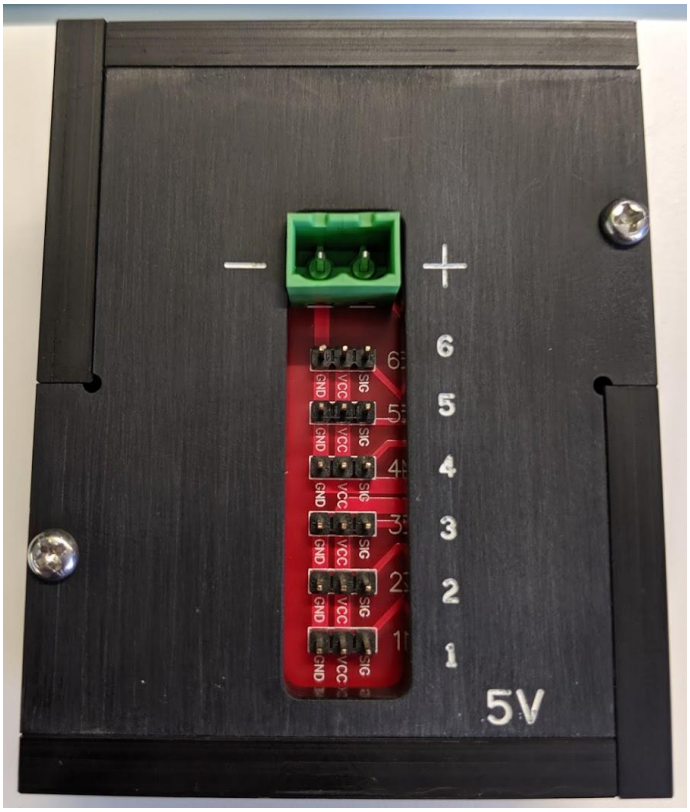


Figure 7: Controller placed in protective enclosure.

Software control from host

Micro-code can be downloaded to the UNO to assign the PWM pins to the motor signals and define the pulse-width ranges for different angles. Arduino micro-code is usually saved as an `.ino` file. The micro-code for servo rotation control of the six, HS-755HB motors is included in Appendix A. The Arduino servo library only supports 0 to 180 rotation, so it is important to normalize the actual rotation of the motor to get movement between 0 and 180 degrees as the HS-755HB has ~200° of throw. Additionally, in practice we have seen the actual pulse-width range of operation for the HS-775HB is 560 to 2500 us rather than the specified 500 to 2500 us.

Lighting control has been added to the Arduino controller to allow for flash tests, save power, and extend the lifetime of the test rigs. Using one of the digital output pins, D12, on the Arduino allows a controllable power supply to be turned ON and OFF.

A conceptual diagram for a fully populated host is shown in figure 8.

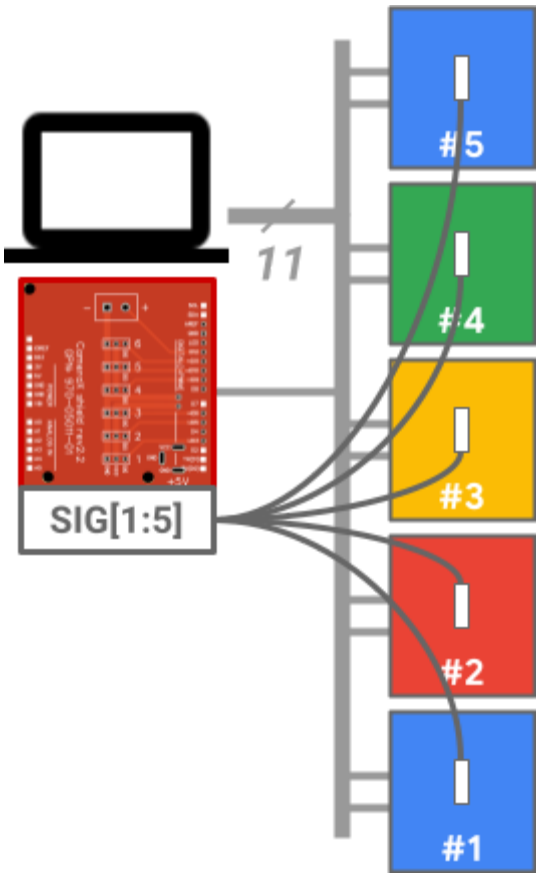


Figure 8: conceptual diagram of fully populated 5x test rig configuration.

Additional BOM items required for lighting control are:

Qty	Part	PN/Link
1	Digitally controllable power strip	https://www.adafruit.com/product/2935
1	22 gauge twisted pair red/black wires 24”	
2	12 V, 36 W UL-listed wall adaptor, 2.1x5.5 mm plug (lights)	
2	3-way, 2.1 mm jack female-male splitter	
6	2.1 mm jack extension cables	https://www.digikey.com/10-02228-ND

Appendix B shows a simple program to turn the lights ON, move a motor between 0, 90, and 180 degrees and back, and turn the lights OFF.

CameraX lab build BOM

The 52-pod CameraX lab switched to the new controllers.
Order tracking spreadsheet

GPN

GPN 1054735

Board revision history

Revision	Description	Notes
rev2.2	D12 for lighting control	Changed to D12 → D13. D13 is overloaded for indicator LED
rev2.1	D13 for lighting control	Simplify to single control and reduce routing by using D13 instead of D2 Also, D0 & D1 overloaded with RX/TX comm
rev2.0	D0, D1, D2 for lighting control	Initial 6x design
rev1.0	No lighting control, 4x motors	

Appendix A

Arduino Micro-code

```
// Import Arduino Servo library
#include <Servo.h>

// Create Servo objects
Servo servo_1;
Servo servo_2;
Servo servo_3;
Servo servo_4;
Servo servo_5;
Servo servo_6;

// Start bits for different items. Note, these much match external script
int servo_start_byte = 255;
int light_start_byte = 254;

// HS-755HB servo datasheet values
int min_pulse_width = 560;   // us
int max_pulse_width = 2500;  // us

// Lighting control channel
int light_channel = 12;  // Arduino D12 pin

// User input for servo and position
int cmd[3];           // raw input from serial buffer, 3 bytes
int servo_num;        // servo number to control
int angle;            // movement angle
int i;                // counter for serial read bytes

void setup() {
  // Attach each Servo object to a digital pin
  servo_1.attach(3, min_pulse_width, max_pulse_width);
  servo_2.attach(5, min_pulse_width, max_pulse_width);
  servo_3.attach(6, min_pulse_width, max_pulse_width);
  servo_4.attach(9, min_pulse_width, max_pulse_width);
  servo_5.attach(10, min_pulse_width, max_pulse_width);
  servo_6.attach(11, min_pulse_width, max_pulse_width);

  // Open the serial connection, 9600 baud
  Serial.begin(9600);

  // Initialize at current position
  servo_1.write(0);
  servo_2.write(0);
```

```
  servo_3.write(0);
  servo_4.write(0);
  servo_5.write(0);
  servo_6.write(0);

  // Create digital output & initialize HIGH
  pinMode(light_channel, OUTPUT);
  digitalWrite(light_channel, HIGH);
}

void loop() {
  if (Serial.available() >= 3) {
    // Read command
    for (i=0; i<3; i++) {
      cmd[i] = Serial.read();
      Serial.write(cmd[i]);
    }
    if (cmd[0] == servo_start_byte) {
      servo_num = cmd[1];
      angle = cmd[2];
      switch (servo_num) {
        case 1:
          servo_1.write(angle);
          break;
        case 2:
          servo_2.write(angle);
          break;
        case 3:
          servo_3.write(angle);
          break;
        case 4:
          servo_4.write(angle);
          break;
        case 5:
          servo_5.write(angle);
          break;
        case 6:
          servo_6.write(angle);
          break;
      }
    }
    else if (cmd[0] == light_start_byte) {
      if (cmd[2] == 0) {
        digitalWrite(light_channel, LOW);
      }
      else if (cmd[2] == 1) {
        digitalWrite(light_channel, HIGH);
      }
```

```
        }
    }
}
```

Appendix B

Simple program to move 0, 90 degrees

```
"""CameraX Arduino box rotator controller.


Usage:
python rotator.py
 --ch <channel number of rotator, by default is 1>
 --dir <Rotate direction(ON: rotate, OFF: return to 0)>
 --debug
"""

import argparse
import codecs
import logging
import struct
import time
import pyudev
import serial

ARDUINO_ANGLE_MAX = 180  # degrees
ARDUINO_BAUD_RATE = 9600
CMD_TEST = [b'\x01', b'\x02', b'\x03']  # test hex command to send to Arduino
CMD_WORD_LENGTH = 3  # bytes
HS755HB_ANGLE_MAX = 202.0  # degrees
NUM_TRYS = 3  # number of tries to get serial port to communicate properly
START_BYTE_SERVO = 255


def get_cmdline_args():
 """Get command line arguements."""
 parser = argparse.ArgumentParser()
 parser.add_argument('--ch', help='--ch [1:6]', default=1, required=True,
                     type=int, choices=[1, 2, 3, 4, 5, 6])
 parser.add_argument('--dir', help='--dir OFF|ON', required=True, type=str,
                     choices=['ON', 'OFF'])
 parser.add_argument('--offset', help='--offset #', default=0, type=int)
 parser.add_argument(
     '--debug', help='--debug', default=False, action='store_true')
 args = parser.parse_args()

 # determine angle for rotation from ON|OFF direction
 angle = 0  # default OFF
 if args.dir == 'ON':
   angle = 90
```

```
  return args.ch, angle, args.offset, args.debug


def serial_port_def(name):
  """Determine the serial port and open.

  Args:
    name: Device to locate (ie. 'Arduino')

  Returns:
    Serial port object
  """
  devices = pyudev.Context()
  for device in devices.list_devices(subsystem='tty', ID_BUS='usb'):
    if name in device['ID_VENDOR']:
      arduino_port = device['DEVNAME']
      continue
  logging.info('Using port %s', arduino_port)
  return serial.Serial(arduino_port, ARDUINO_BAUD_RATE, timeout=1)


def read_command(port):
  """Read back command from serial port."""
  cmd = []
  for _ in range(CMD_WORD_LENGTH):
    cmd.append(port.read())
  return cmd


def send_command(port, cmd):
  """Send command to serial port."""
  for i in range(CMD_WORD_LENGTH):
    port.write(cmd[i])
  time.sleep(2.0 * CMD_WORD_LENGTH / ARDUINO_BAUD_RATE)  # round trip
  return read_command(port)


def establish_serial_comm(port):
  """Establish connection with serial port."""

  logging.info('Establishing communication with %s', port.name)
  attempts = 0
  hex_test = convert_to_hex(CMD_TEST)
  logging.debug(' test tx: %s %s %s', hex_test[0], hex_test[1], hex_test[2])
  while attempts < NUM_TRYS:
    cmd_read = send_command(port, CMD_TEST)
    hex_read = convert_to_hex(cmd_read)
    logging.debug(' test rx: %s %s %s', hex_read[0], hex_read[1], hex_read[2])
```

```
    if cmd_read != CMD_TEST:
      attempts += 1
    else:
      logging.debug('Arduino comm established after %d attempt(s)', attempts)
      break


def convert_to_hex(cmd):
 # compatible with both python 2 and python 3
 return [('%0.2x' % int(codecs.encode(x, 'hex_codec'), 16) if x else '--')
         for x in cmd]


def rotate_servo(servo, angle, port):
 """Rotate servo to the specified angle.

 Args:
   servo: int; servo to rotate [1,2,3,4,5,6]
   angle: int; servo angle to move to
   port: object; serial port

 Returns:
   None
 """

 err_msg = 'Servo angle %.1f must be between 0 and %d.' % (
     angle, ARDUINO_ANGLE_MAX)
 if angle < 0:
   logging.error('%s', err_msg)
   angle = 0
 elif angle > ARDUINO_ANGLE_MAX:
   logging.error('%s', err_msg)
   angle = ARDUINO_ANGLE_MAX
 cmd = [struct.pack('B', i) for i in [START_BYTE_SERVO, servo, angle]]
 hex_cmd = convert_to_hex(cmd)
 logging.debug(' tx: %s %s %s', hex_cmd[0], hex_cmd[1], hex_cmd[2])
 cmd_read = send_command(port, cmd)
 hex_read = convert_to_hex(cmd_read)
 logging.debug(' rx: %s %s %s', hex_read[0], hex_read[1], hex_read[2])


def main():
 # get channel, rotate, offset, debug from command line
 servo, angle, offset, debug = get_cmdline_args()

 # set logging level
 if debug:
   logging.basicConfig(level=logging.DEBUG)
```

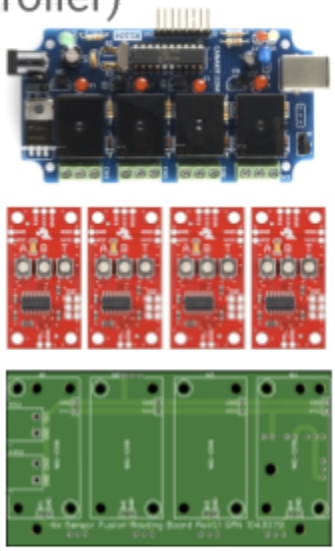
```
 else:
   logging.basicConfig(level=logging.INFO)

 # initialize port
 serial_port = serial_port_def('Arduino')

 # send test commands to Arduino until command returns properly
 establish_serial_comm(serial_port)

 # rotate servo to angle
 angle_norm = int(
     round((angle - offset) * ARDUINO_ANGLE_MAX / HS755HB_ANGLE_MAX, 0))
 logging.info('Moving servo %d to position %d', servo, angle)
 rotate_servo(servo, angle_norm, serial_port)


if __name__ == '__main__':

 main()
```

Appendix C

BOM Cost Reduction

rev1 BOM (sensor_fusion rack-mount controller)

Part	#	$
CanaKit relay controller UK1104	1	67.20
SparkFun servo trigger WIG13118	4	17.95
Texas Instruments regulator LM1084	1	2.50
Custom routing board	1	2.00
		143.50

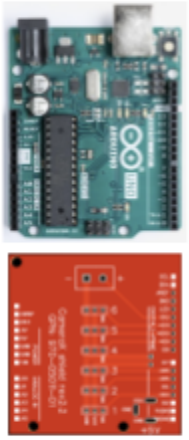


Other notes

- CanaKit requires rework to connect to routing board
- CameraX lab uses these now (+2 hot-swap spares)

rev2 BOM

Part	$
Arduino UNO	22.00
Custom routing board	1.00
	23.00



Other benefits

- 5V external power for motor → no on-board regulator
- Easier assembly → no board rework, just populate board
- 6 PWL outputs → 9 hosts (instead of 13) for 52 test rigs