Motor Control Workshop Content:

Low-Level Servo Control

Provided that i2c is enabled in PI-configuration, you may run code that uses the i2c communication which is the SMBus module. To control the servos first import that module and declare a bus connection.

- import smbus
- bus = smbus.SMBus(1)

Then provide the part address for the connection to go to which is 0x40 by default:

- addr = 0x40

Then initialize the settings for the communication by enabling the PWM chip and tell it to automatically increment addresses after a write (that lets us do single-operation multi-byte writes)

- bus.write_byte_data(addr, 0, 0x20)
- bus.write_byte_data(addr, 0xfe, 0x1e)

With this initialized, now the servos can be told to move to certain positions via Pulse Width Modulation (a method of changing the power delivered to a motor by changing the average voltage in the pulses). The yellow cable of the servo connector carries the control signal, used to tell the motor where to go. This control signal is a specific type of pulse train. Servo motors get their control signal from that pulse width. We can write commands to the servo that tell how big that pulse is, for example:

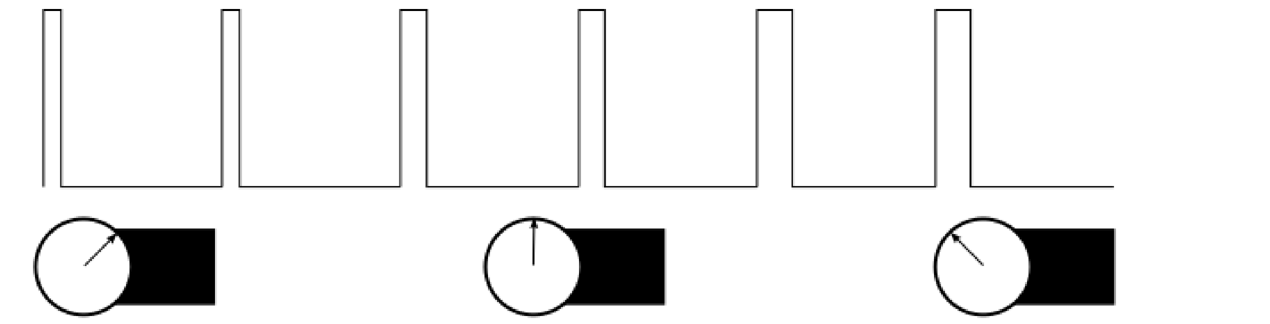
- bus.write_word_data(addr, 0x06, 0)
- bus.write_word_data(addr, 0x08, 1250)

The first write is to the “start time” register for channel 0 (indicated by using 0x06). By default, the PWM frequency of the chip is 200Hz, or one pulse every 5ms. The start time register determines when the pulse goes high in the 5ms cycle. All channels are synchronized to that cycle. Generally, this should be written to 0 i.e. the start of the period should be high. This only needs to be done once.

The second write is to the “stop time” register, and it controls when the pulse should go low (indicated by using 0x08 for channel 0). The range for this value is from 0 to 4095, and each count represents one slice of that 5ms period (5ms/4095), or about 1.2us. Thus, the value of 1250 written above represents about 1.5ms of high time per 5ms period.

Generally speaking, a pulse width of 1.5ms yields a “neutral” position, halfway between the extremes of the motor’s range. 1.0ms yields approximately 45 degrees off center, and 2.0ms yields -45 degrees off center. In practice, those values may be slightly more or less than 45 degrees, and the motor may be capable of slightly more or less than 45 degrees of motion in either direction. Common servos rotate over a range of 90° as the pulses vary between 1 and 2ms – they should be at the center of their mechanical range when the pulse is 1.5ms: https://learn.sparkfun.com/tutorials/hobby-servo-tutorial#servo-motor-background

To achieve 1ms pulses, use a stop time of 836, likewise, to get 2ms pulses use a stop time of 1664.


1 millisecond pulse = full clockwise
1.5 millisecond pulse = centered
2 millisecond pulse = full counterclockwise

Using the module time to pause the code so that the servo has time to adjust to its position:

- time.sleep(2)   # pause for two seconds
- bus.write_word_data(addr, 0x08, 836) # chl 0 end time = 1.0ms
- time.sleep(2)
- bus.write_word_data(addr, 0x08, 1664) # chl 0 end time = 2.0ms

Essentially, the servo position is controlled by the stop time of the pulse of the PWM which is a value that ranges from 0 (no power) to 4095 (Full power). Keeping it safe, the range used is from 836 to 1664 which allows a good 90 degrees of movement (acceptable for the arm to pick and place objects in its specific workspace).

See Appendix A for the start and stop addresses for using other channels on the PI hat.

For channel 0 in the example, the start address is 0x06 and the stop address is 0x08.

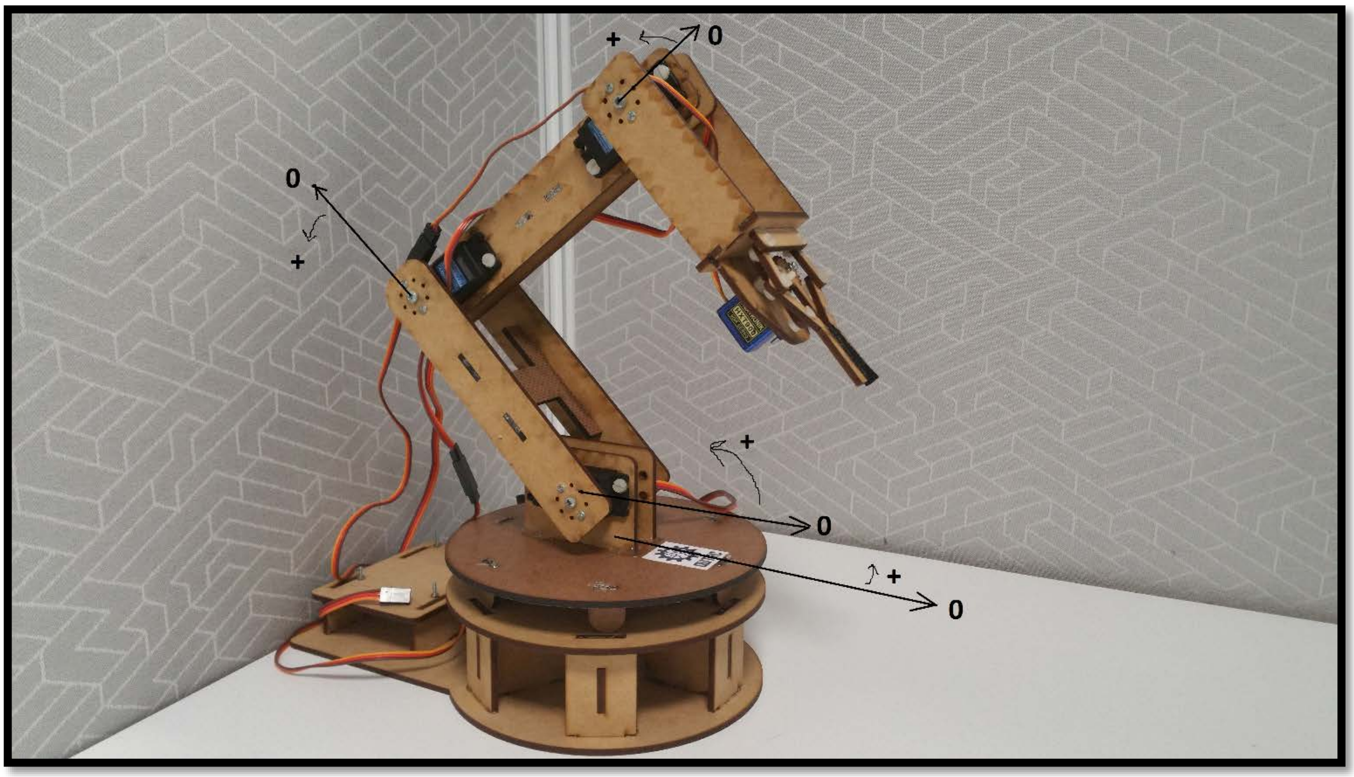
High Level Control

The low level control showed that the position of the servo can by controlled by PWM pulses governed by integers from the range of 0 to 4095 with a mechanical working range of 836 to 1664. For the interest of understanding the kinematics of the arm, it is necessary to map that PWM to physical degrees or radians of rotation. For that, calibration is done where one should measure the angle output from each joint that undergoes the PWM values of 836, 1250 and 1664. The relationship is proportional and hence a linear relationship is required to map PWM to angle output.

Using the calibration data, one can model the servo as a class object in python that has variables like joint limits and functions that can move the servo to a certain degrees or radians or read the current position of the joint. The functions for the PI servo class is in the Robotarm_ServoModule.py file in the workshop folder.

Measuring Convention of the Joints of the Arm:

To measure the joints consistently for each arm, the standard for placing the origin for the angles is proposed based on the right hand rule about the axis of rotation and the coordinate frames later on. To get the idea, the robot in the picture has the revolute base at 0, the next joint is at 135, the joint above that is at -100 and the joint above that one is at -90 degrees.



For the Robot arm, generally channels 0 to 4 are used for the whole arm. An easy pattern to find the addresses is that each hexadecimal increments by 4 for every channel:

Example:

Ch.0 start = 0x06,   Ch.1 start = 0x06 + 4*1 = 0x0A,   Ch.2 start = 0x06 + 4*2 = 0x0E

Ch.0 stop = 0x08,   Ch.1 stop = 0x08 + 4*1 = 0x0C,   Ch.2 stop = 0x08 + 4*2 = 0x10

Channel #	Start Address	Stop Address
Ch. 0	0x06	0x08
Ch. 1	0x0A	0x0C
Ch. 2	0x0E	0x10
Ch. 3	0x12	0x14
Ch. 4	0x16	0x18
Ch. 5	0x1A	0x1C
Ch. 6	0x1E	0x20
Ch. 7	0x22	0x24
Ch. 8	0x26	0x28
Ch. 9	0x2A	0x2C
Ch. 10	0x2E	0x30
Ch. 11	0x32	0x34
Ch. 12	0x36	0x38
Ch. 13	0x3A	0x3C
Ch. 14	0x3E	0x40
Ch. 15	0x42	0x44