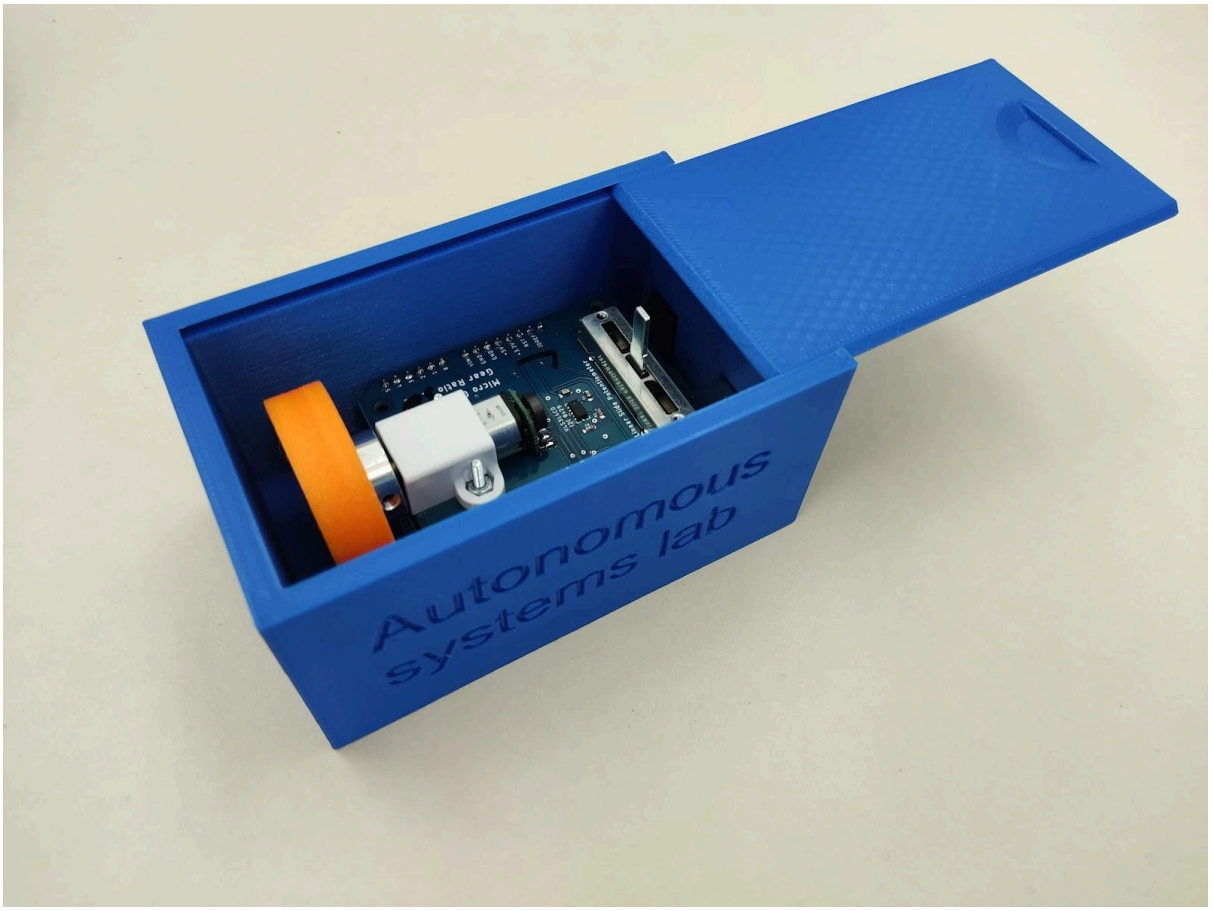
Home kit TAU-ARD V01



1. Introduction

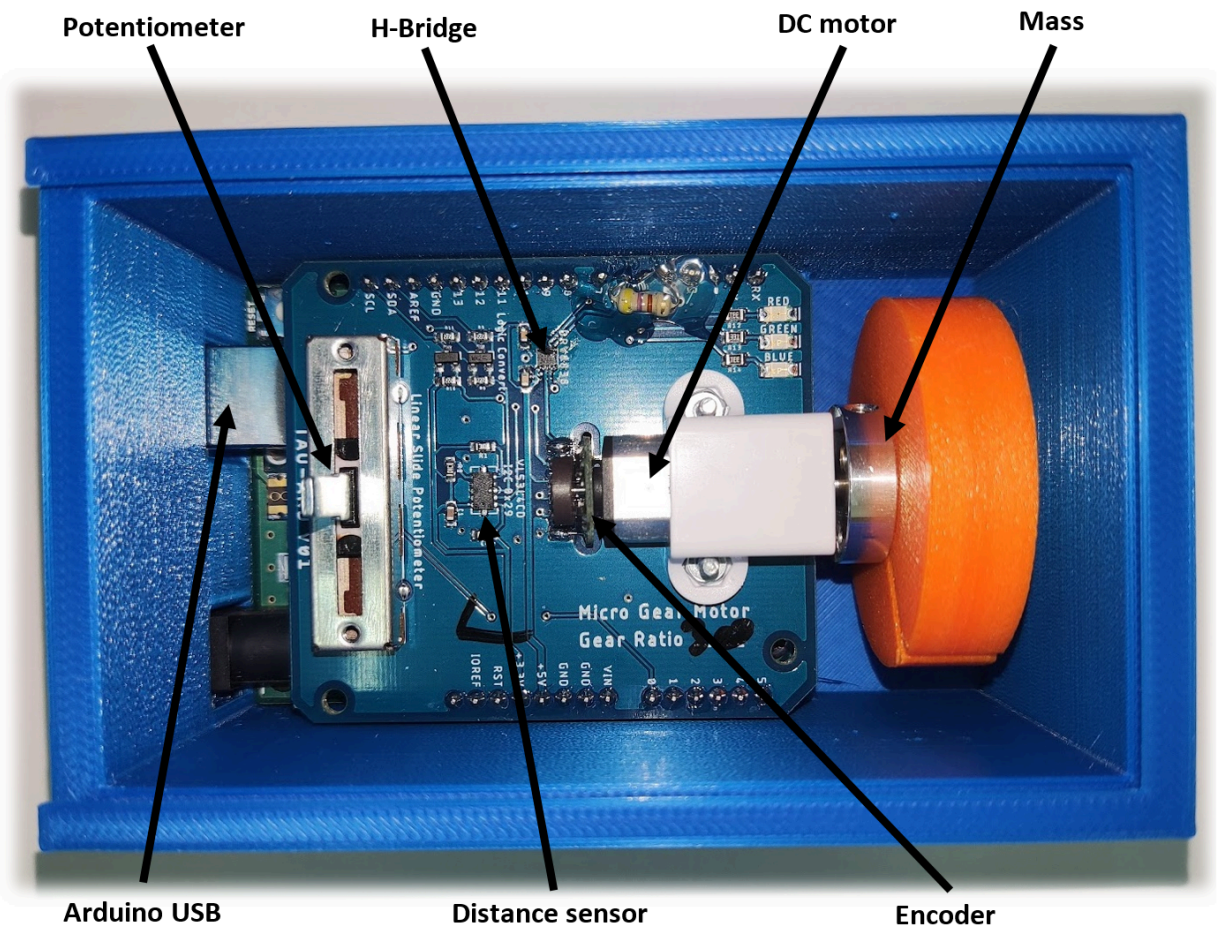
Modern control systems are realised using a Microcontroller Unit (MCU) and implemented in code. Since a MCU is a discrete system, as such the control implementation is also a discrete control system. In this home experiment setup, you will get a taste of working with a discrete system, reading sensor data and implementing a simple motor control, using an Arduino development board and the Arduino Integrated Development Environment (IDE).

2. System overview

The home kit setup consists of an Arduino Uno development board with a custom carrier shield with a motor driver, a DC motor including a gearbox, encoder and a rotation mass. In addition, the shield includes some LEDs, a potentiometer and a digital distance sensor for user interaction. The motor control is implemented using an Arduino development environment in C++. For thorough description of the setup concepts see the **literature review document**.

System components:

Arduino Uno Micro Controller Unit (MCU):
https://store.arduino.cc/arduino-uno-rev3
DC Motor including Gear box 1:100, 1:50, 1:30 etc.:
https://www.pololu.com/product/2202
Magnetic Encoder with 3PPR (Pulses Per Revolution):
https://www.pololu.com/product/3081
Rotating mass Flywheel:
https://www.pololu.com/product/1996
PC with an Arduino IDE Connected over USB cable:
https://www.arduino.cc/en/Main/Software
https://www.4project.co.il/product/a2b-standard-usb-cable-1.8m
3D Printed Case:
https://a360.co/3ynSQNA
Custom adapter shield
https://a360.co/3QQH5q1



3. Getting started with Arduino / Arduino IDE

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino uses C++ programming language with an extended library based on the Wiring Project, the Arduino Software (IDE) is based on Processing. Over the years Arduino has been the brain of thousands of projects, from everyday objects to complex scientific instruments. A worldwide community of makers - students, hobbyists, artists, programmers, and professionals - has gathered around this open-source platform, their contributions have added up to an incredible amount of accessible knowledge that can be of great help to novices and experts alike.

Setting up Arduino Integrated Environment (IDE)

Head on to the Arduino site and install either the classic IDE 1.xx or the new improved version 2.xx. This tutorial will present the examples as an IDE 1.xx screenshots.

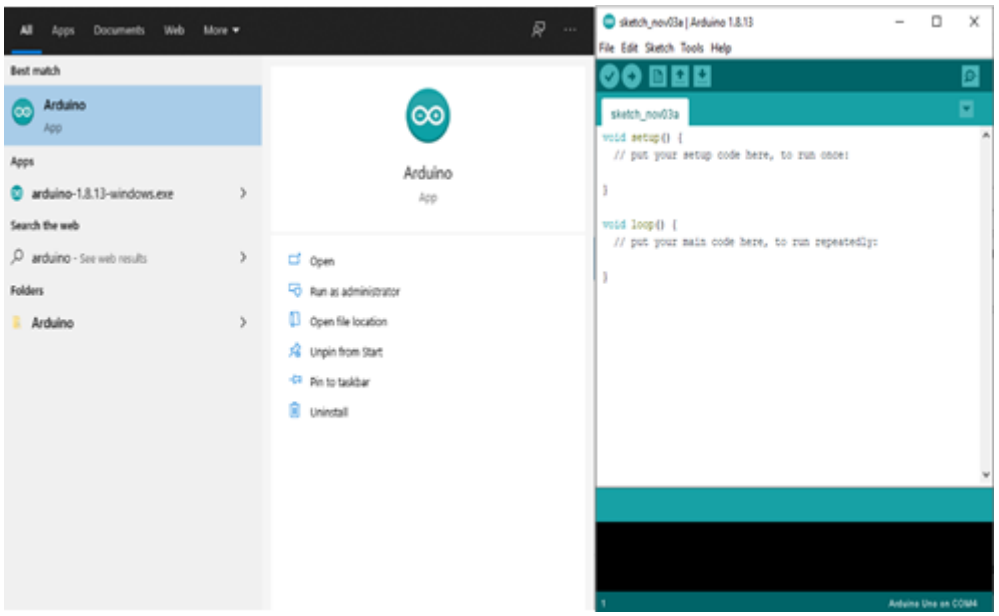
https://www.arduino.cc/en/software

you may consult the getting started guide for some tips on how to get up and running using Arduino.

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

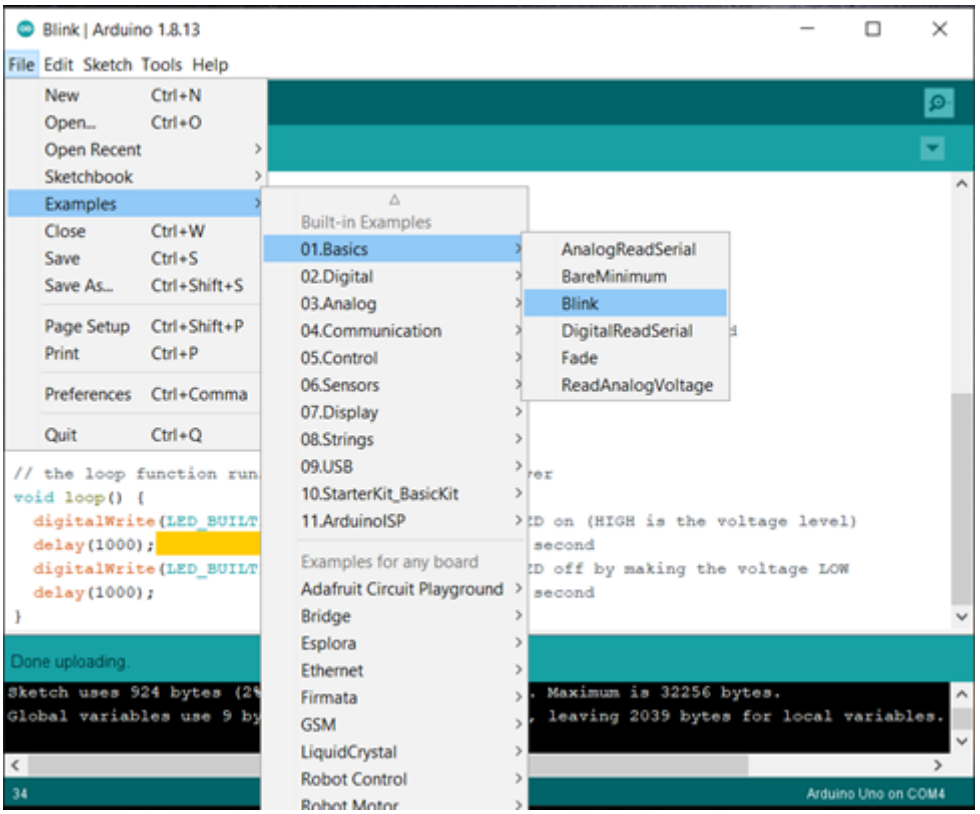
Running a simple example

Open Arduino IDE, either use the desktop icon, or search for the application.



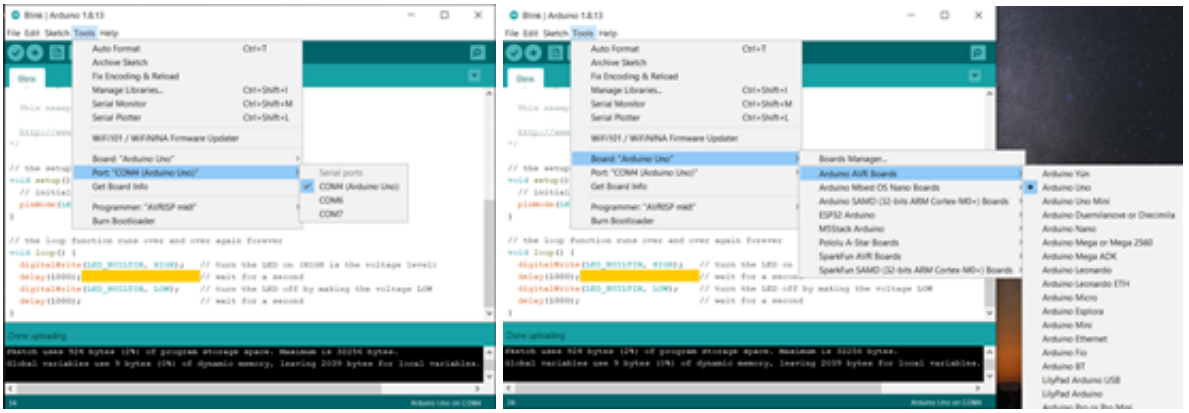
Connect the USB cable to the Home Kit Box. Since the micro controller implements the last code that has been uploaded, you may see the motor start spinning upon voltage application. When you finish working with the setup, disconnect the USB cable.

We will begin by opening a simple blink example and uploading the code to the microcontroller. Select: File→Examples→Basics→Blink.

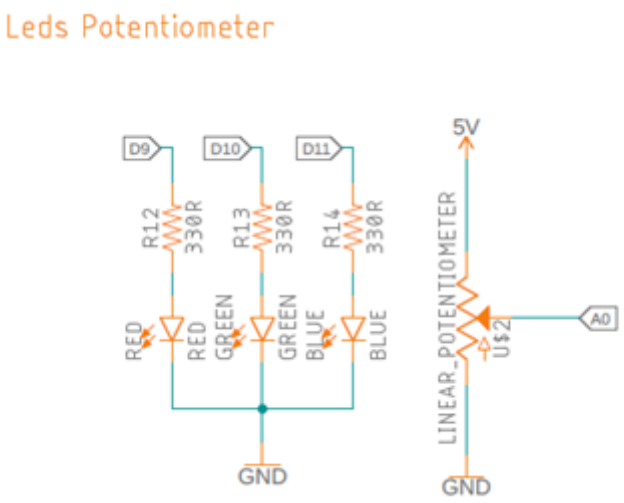


Connect to the Arduino Uno microcontroller and make sure the correct port is selected. The COM number may change between computers.

- Select: Tools→Board→Arduino Uno,
- Select: Tools→Port→COMX (Arduino Uno),



Before uploading the code, we will modify it to blink one of the LEDs present on the shield by changing the LED_BUITLIN parameter to the pin number connected to the Arduino. For example, pin number: 9



Upload the modified code to the microcontroller by pressing the upload button →. If everything is successful then you should see the “Done uploading” message and the led will start blinking.

Tasks:

- Blink with another led
- Modify blinking time
- Create sequenced blinking effects.

4. Working with the home kit setup

The chapter is dedicated to working with the kit`s hardware, Serial communication, reading an analog value of a potentiometer. Running a DC motor and reading the connected encoder. Working with I2C communication with a distance sensor.

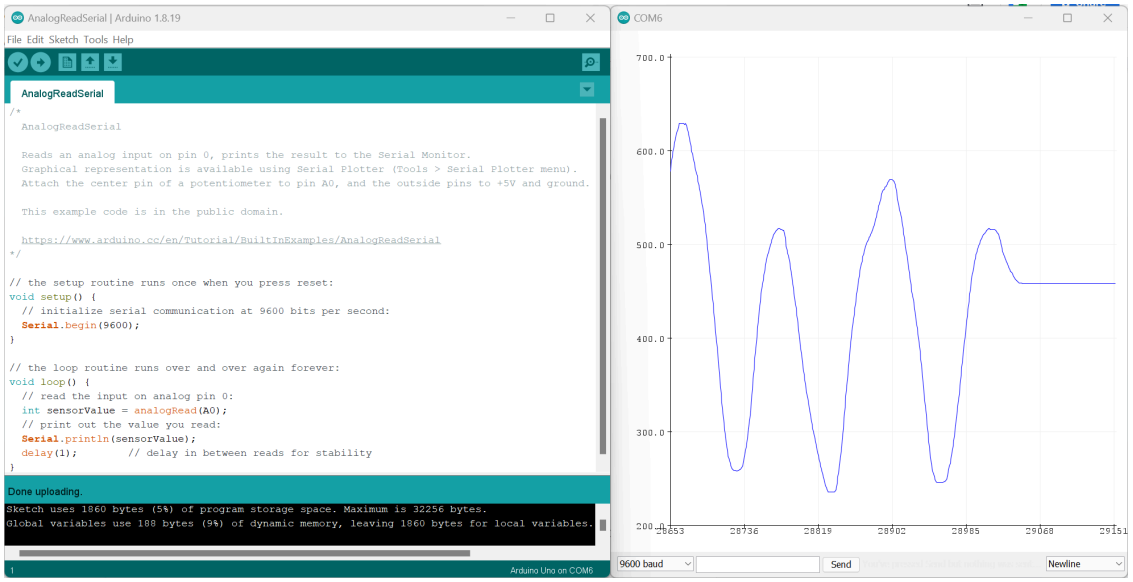
Serial Communication & Potentiometer

Using the arduino IDE open the example AnalogReadSerial located under: File→Examples→Basics→AnalogReadSerial

Upload the example to the microcontroller and notice the value changes using the serial monitor and serial plotter located under:

- Tools→Serial Monitor
- Tools→Serial Plotter

Note: Verify that the baud rate is set as in the example code. 9600



Tasks:

- Notice that the value ranges from 0 - 1023 which corresponds to 10 bit, calculate the required conversion to present the measurements in volts and update the code accordingly.
- Review the Fade example located under: File→Examples→Basics→Fade and update the code to change the LEDs brightness based on the readings of the potentiometer

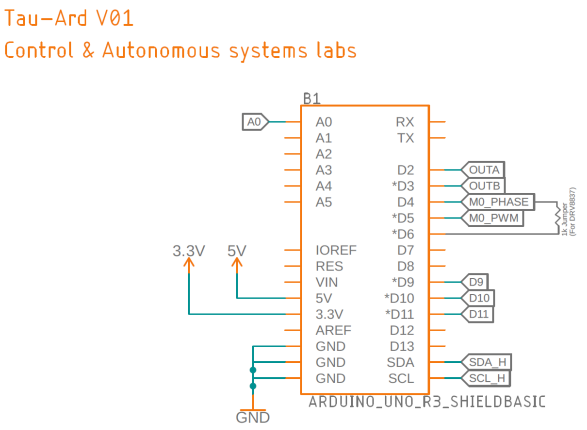
DC motor

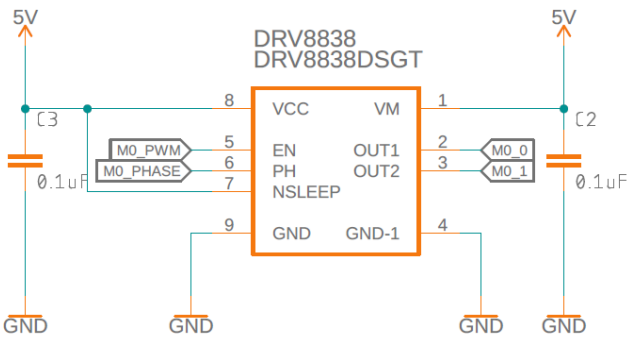
Due to the chip shortage in the industry some of the boards are assembled with the DRV8837 instead of the DRV8838 dc motor driver. You may identify it by the presence of a small resistor soldered between pin 4 and pin 6 on the board.

The reason for the resistor is due to different requirements for controlling the driver. While DRV8838 requires only one PWM pin and one logic pin, DRV8837 Requires 2 PWM pins.

M0_PWM is connected to pin D5

M0_PHASE is connected to pin D4, in case a resistor is soldered to the board M0_PHASE is connected to D6 for the additional required PWM pin.





The DRV8837 device is controlled using a PWM input interface, also called an IN-IN interface. Each output is controlled by a corresponding input pin.

Table 7-1 shows the logic for the DRV8837 device.

Table 7-1. DRV8837 Device Logic

nSLEEP	IN1	IN2	OUT1	OUT2	FUNCTION (DC MOTOR)
0	X	X	Z	Z	Coast
1	0	0	Z	Z	Coast
1	0	1	L	H	Reverse
1	1	0	H	L	Forward
1	1	1	L	L	Brake

The DRV8838 device is controlled using a PHASE/ENABLE interface. This interface uses one pin to control the H-bridge current direction, and one pin to enable or disable the H-bridge.

Table 7-2 shows the logic for the DRV8838.

Table 7-2. DRV8838 Device Logic

nSLEEP	PH	EN	OUT1	OUT2	FUNCTION (DC MOTOR)
0	X	X	Z	Z	Coast
1	X	0	L	L	Brake
1	1	1	L	H	Reverse
1	0	1	H	L	Forward

Tasks:

- Using the supplied schematics, chip datasheet and based on the driver type: implement the required logic to run the motor in CCW direction, Halt, and CW direction for 5 seconds for each motion section.
- Run the motor based on the potentiometer position i.e. Middle-Motor stops, Left-rotates CCW, Right-rotates CW. The more off centre the potentiometer the faster the rotation.
- Update the LED`s corresponding to the motor rotation.

Hint:

In case of DRV8837 use two PWM outputs D5 (IN1) and D6 (IN2), each time one of them set to 0 depending on the required rotation direction.

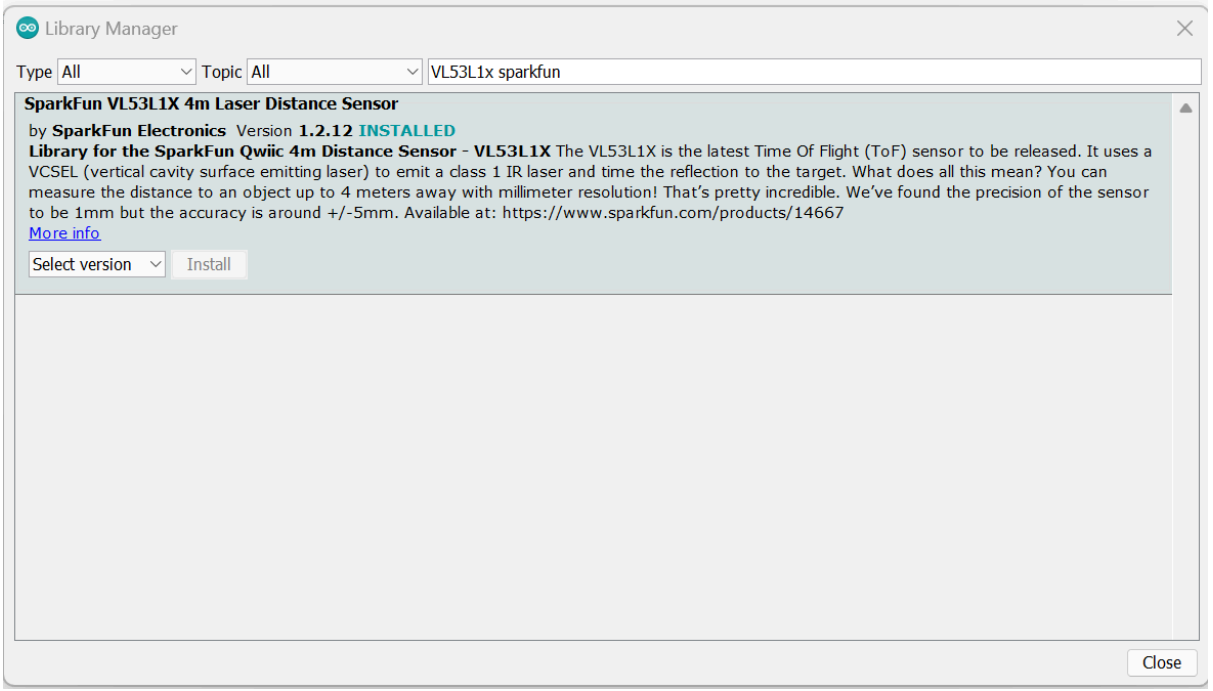
In case of DRV8838 use one digital pin D4 for the motor direction (PH), and the other for the PWM output D5 (EN)

Use analogWrite command to set the PWM duty cycle.

Distance sensor - VL53L4CD

The VL53L4CD is a compact Time Of Flight (ToF) sensor, capable of detecting an object at up to 1.3m. A thorough review and a tutorial for the sensor is available by sparkfun. We will use Sparkfun Library available on Github and can be added through the arduino IDE

- Open Library Manager Tools→Manage Libraries
- Search for VL53L1x sparkfun library which is also compatible with the sensor
- Install the library



- Open Example1_ReadDistance located under File→Examples→SparkFun_VL53L1X…→Example1_ReadDistance
- Upload the example and open the serial monitor. Make sure to update the baud rate to the baud rate set in code. (115200)

Tasks:

- Use the Serial Plotter to plot the sensor data.
- Run the motor based on the measured distance, the farther the object the faster the rotation
- Review the example code Example6_ArduinoPlotterOutput File→Examples→SparkFun_VL53L1X…→Example6_ArduinoPlotterOutput. Modify the code for non blocking implementation changing the while statement to if and implementing measurement when the condition for checkForDataReady() is met.

Magnetic Encoder - Interrupts

There are many types of encoders, the Home kit includes a magnetic encoder with 3 pulses per rotation and quadrature resolution of 12 increments per revolution of the motor shaft. Some microcontrollers include dedicated hardware to count the pulses when such a hardware is absent it is possible to count the pulses using an external interrupt event, more about interrupts is available at: Arduino - External Interrupts.

Open blank arduino sketch and add the following code for a basic example of reading and encoder with an external interrupt:

```
//Encoder Example

// Define Pins
#define ENCODER_PINA 2
#define ENCODER_PINB 3

// encoder variables
volatile int encoderCounts = 0;

// Encoder ISR functions - Interupt Service Routine
void encoderA();
void encoderB();

void setup() {
  // initialize serial communication at 115200 bits per second:
  Serial.begin (115200);

  // initialize encoder, attache ISR functions
  pinMode(ENCODER_PINA, INPUT);
  pinMode(ENCODER_PINB, INPUT);
   // Attached interrupt to encoder pins
  attachInterrupt(digitalPinToInterrupt(ENCODER_PINA), encoderA, CHANGE);
  attachInterrupt(digitalPinToInterrupt(ENCODER_PINB), encoderB, CHANGE);

  Serial.print("Encoder_Value");
}

void loop() {
  // print encoder position
  Serial.println(encoderCounts);
  delay(100);
}

// EncoderA ISR
void encoderA() {
  // look for a low-to-high on channel B
  if (digitalRead(ENCODER_PINA) == HIGH) {
    // check channel A to see which way encoder is turning
    digitalRead(ENCODER_PINB) ? encoderCounts++ : encoderCounts--;
  }else{
    // check channel A to see which way encoder is turning
    digitalRead(ENCODER_PINB) ? encoderCounts-- : encoderCounts++;
  }
} // End EncoderA ISR

// EncoderB ISR
void encoderB() {
  // look for a low-to-high on channel B
  if (digitalRead(ENCODER_PINB) == HIGH) {
    // check channel A to see which way encoder is turning
    digitalRead(ENCODER_PINA) ? encoderCounts-- : encoderCounts++;
  }else{
    // check channel A to see which way encoder is turning
    digitalRead(ENCODER_PINA) ? encoderCounts++ : encoderCounts--;
  }
} // End EncoderB ISR
```

Tasks:

- Review the code and check what it does.
- Manually rotate the wheel one rotation, based on the encoder reading and assuming that the code counts 12 pulses per revolution of the motor shaft, what is the gear ratio of the motor.
- Adjust the code to convert the measured value to the number of rotation of the wheel.
- Using millis(); or micros(); commands, measure the time between readings and convert the position to velocity in RPM units
- Add the required code to rotate the motor using a potentiometer and plot the measured velocity while changing the potentiometer.

Hint: Make sure to properly convert variable types i.e. int → float when necessary by using commands such as float() , int().

Implement using the method of reading the change in number of pulses in a period of time i.e. such as a main loop with a delay of 100 milliseconds, calculate the number of pulses that are done in 100 milliseconds.

Consider implementing delay with millis() function instead of using the Delay function; Blink Without Delay

5. Arduino serial parser

The serial object in arduino is used for the communication between an Arduino board and a computer or other devices. All Arduino boards have at least one serial port (also known as a UART or USART), and some have several. In order to extract meaningful information out of the USART stream it is required to implement some sort of data parsing mechanism. Arduino provides several examples and objects for that purpose such as the Serial object. In addition it is useful to use the String object for collecting the data and processing it.

Several built in example for handling and parsing serial communication:

- Dimmer
- Read ASCII String
- SerialEvent
- StringIndexOf
- StringSubstring
- StringToInt
- StringToFloat

Tasks:

- Implement a simple parser which receives an ascii string of numbers terminated with '\n' converts the string to a number and updates the LED's brightness accordingly.
- Implement an advanced parser which receives a string terminated with '\n', separates the string into substrings based on a delimiter such as ',' and converts each sub string into an integer. Use the String class for the operations. Do not use Serial.parseInt(); function. Modify the brightness of all three leds using this method.

6. Close loop implementations

Control systems are an integral part of an autonomous system implementation, in this secretion several close loop implementations will be implemented. Position and velocity control close loop based on an encoder feedback. User interaction with a close loop implementation through an additional sensor such as a potentiometer or a distance sensor. A basic PID implementation. For a motor control in is often sufficient to implement a simple PI controller.

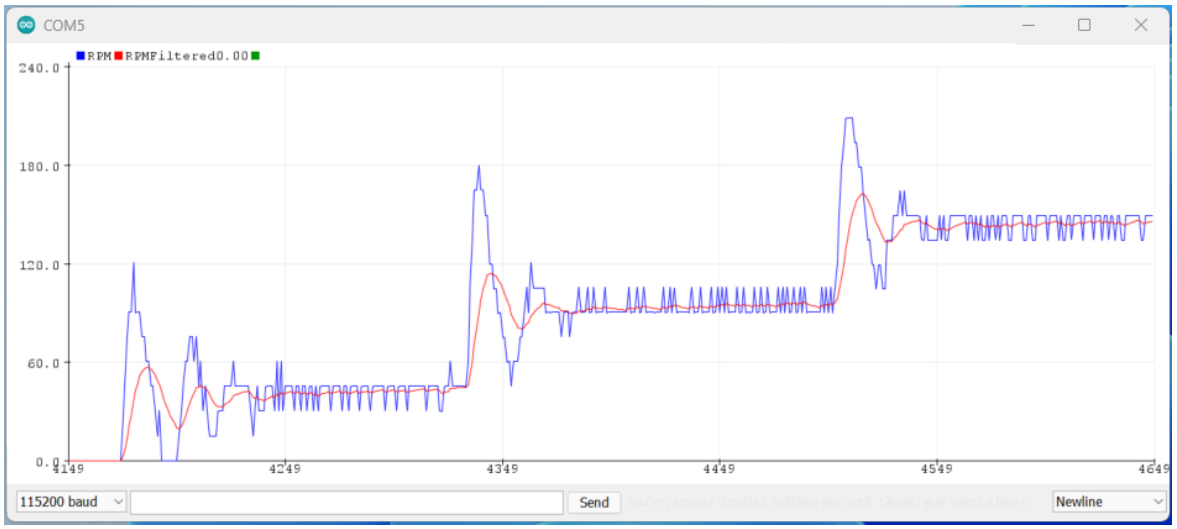
Tasks:

Position control close loop

Position control with an encoder feedback is one of the most basic implementations of a control loop. Using the previous examples / tasks implement a position control loop based on an encoder measured position. Using the serial terminal send the desired position of the rotating mass and verify that it has turned correctly i.e 1 revolution. Implement a PI controller to close the loop and plot a step response. It is desired that the controller will be implemented at a frequency of ~ 100Hz.

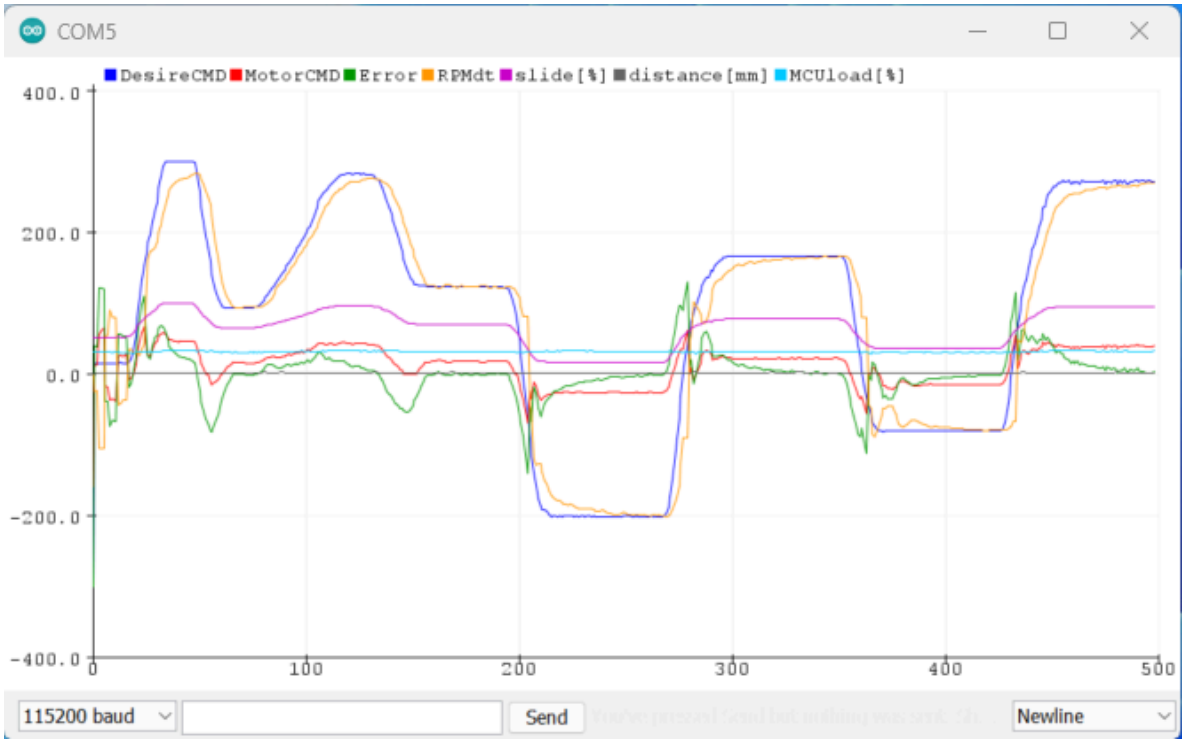
Velocity control close loop

Velocity control with an encoder feedback is one of the most basic implementations of a control loop. Using the previous examples / tasks implementing a velocity control loop based on an encoder based measured velocity, it is suggested to implement a simple LPF to filter the measured velocity. Using the serial terminal send the desired velocity of the rotating mass and verify that it is rotating correctly i.e 60 RPM or 1 revolution per second. Implement a PI controller to close the loop and plot a step response. It is desired that the controller will be implemented at a frequency of ~ 100Hz.



Potentiometer / Distance sensor feedback velocity control

Human robot interaction is of main importance in any autonomous system. Update the code to change the desired velocity of the motor based on the Distance sensor or the Potentiometer readings.



Notes:

Use matlab or python for plotting the responses properly.

Present a block diagram for the close loop implementation

7. Python arduino communication

Setting up the environment

Set up the python environment either using Anaconda , PyCharm or any other environment you are familiar / comfortable with. Add the following libraries to the python environment: pyserial , matplotlib and its dependencies. Add the ArduinoJson library to the arduino ide.

Upload the arduino sketch and run the python script - remember to update the communication port variable based on what is set in your pc / environment.

Python / Arduino codes

Arduino Code

```
#include <ArduinoJson.h>

const int SENSOR_1_PIN = A0;
const int UPDATE_INTERVAL_MS = 10;

String inputString = "";         // a String to hold incoming data
bool stringComplete = false;     // whether the string is complete

void setup() {
  Serial.begin(115200);
  while (!Serial) continue; // Wait for serial connection
  inputString.reserve(200);
}

void loop() {
  static unsigned long last_update_time = 0;

  unsigned long current_time = millis();
  if (current_time - last_update_time >= UPDATE_INTERVAL_MS) {
    last_update_time = current_time;

    int sensor1_value = analogRead(SENSOR_1_PIN);

    StaticJsonDocument<64> doc;
    doc["sensor1"] = sensor1_value;
    serializeJson(doc, Serial);
    Serial.println();
  }

  processIncomingString();

}

void processIncomingString() {
  if (stringComplete) {
    StaticJsonDocument<64> incoming_doc;
    DeserializationError error = deserializeJson(incoming_doc, inputString);
    if (!error) {
      if (incoming_doc.containsKey("command")) {
        String command = incoming_doc["command"].as<String>();
        if (command == "reset") {
          Serial.println("{'reset': 0}");
        }
      }
    }
    inputString = "";
```

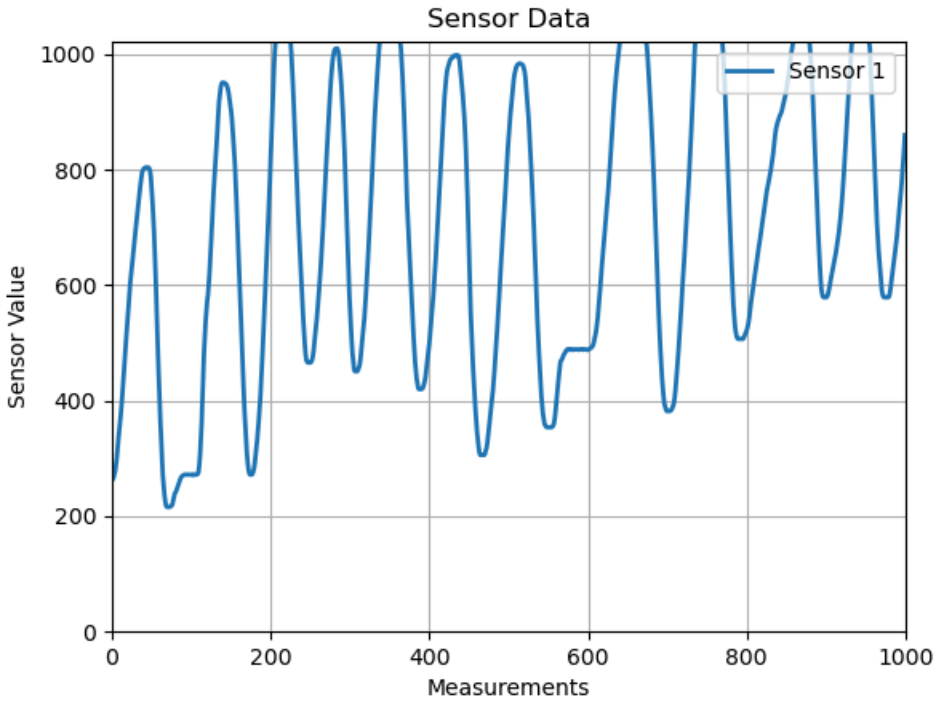
```
    stringComplete = false;
  }
}

void serialEvent() {
  while (Serial.available()) {
    char inChar = (char)Serial.read();
    inputString += inChar;
    if (inChar == '\n') {
      stringComplete = true;
    }
  }
}
```

Python Code

```
import serial
import threading
import json
import time
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from collections import deque

# Define ArduinoCommunication class
class ArduinoCommunication:
    def __init__(self, port='COM5', baudrate=115200, data_filename='sensor_data.txt'):
        self.serial = serial.Serial(port, baudrate)
        self.stop_receive_thread = threading.Event()
        self.receive_thread = threading.Thread(target=self.receive_message_thread, daemon=True)
        self.data_callback = None
        self.receive_thread.start()
        self.data_file = open(data_filename, 'a')

    def __enter__(self):
        return self

    def __exit__(self, exc_type, exc_value, traceback):
        self.close()

    def send_message(self, message):
        message_bytes = message.encode('utf-8')
        self.serial.write(message_bytes)

    def receive_message_thread(self):
        while not self.stop_receive_thread.is_set():
            if self.serial.in_waiting > 0:
                message = self.serial.readline().decode('utf-8').rstrip()
                try:
                    data = json.loads(message)
                    self.save_to_file(message)
                    if self.data_callback:
                        self.data_callback(data)
                except json.JSONDecodeError:
                    print('Error decoding message:', message)

    def close(self):
        self.stop_receive_thread.set()
        self.receive_thread.join()
        self.serial.close()
        self.data_file.close()

    def save_to_file(self, message):
        self.data_file.write(message + '\n')

# Define SensorDataPlot class
class SensorDataPlot:
```

```
    def __init__(self, maxlen=1000):
        self.sensor_data = deque(maxlen=maxlen)
        self.fig, self.ax = plt.subplots()
        self.line, = self.ax.plot([], [], lw=2, label='Sensor 1')

    def handle_data(self, data):
        sensor_value = data.get('sensor1')
        if sensor_value is not None:
            self.sensor_data.append(sensor_value)

    def update_plot(self, num):
        self.line.set_data(range(len(self.sensor_data)), self.sensor_data)
        return self.line,

    def plot(self):
        self.ax.set_xlim(0, self.sensor_data.maxlen)
        self.ax.set_ylim(0, 1023)
        self.ax.set_title('Sensor Data')
        self.ax.set_xlabel('Measurements')
        self.ax.set_ylabel('Sensor Value')
        self.ax.legend(loc='upper right')
        self.ax.grid()  # Add grid to the plot
        ani = animation.FuncAnimation(self.fig, self.update_plot, blit=True, interval=100, repeat=True)
        plt.show()

def main():
    plotter = SensorDataPlot()

    with ArduinoCommunication() as ser:
        ser.data_callback = plotter.handle_data
        plotter.plot()

if __name__ == "__main__":
    main()
```

If everything has been set correctly you should the following plot of the potentiometer values, in addition you will get a recorded file for the received messages, under “sensor_data.txt”



Tasks:

- Review the supplied codes and explain how they work.
- Prepare a block diagram for the supplied codes.
- Adjust the codes to plot the distance sensor, don`t forget to update the title and the axis labels / axis limits.

Notes:

You can use this method for all your needed plots for the report.

Home Kit experiment Tau-Ard V01 shield Schematics: