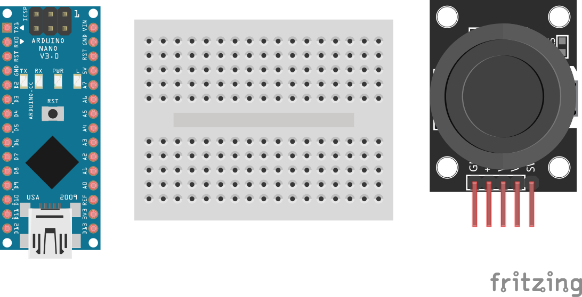
Einheit 1: Joystick

 Materialien

* 1x Arduino Nano mit Verbindungskabel
* 1x mittlere Breadboard
* 1x Joystickmodul
* Jumperkabel weiblich/männlich

Abbildung Materialien

Ein Joystick ist ein Sensor, der die Rotation des Griffs in den zwei Achsen (x-Achse/y-Achse) auslesen kann.

In dieser Einheit wird ein sogenannter analoger Joystick verwendet, welcher die Auslenkung des Griffs mithilfe von zwei Potentiometern ausliest. Ein Potentiometer ist ein elektronisches Bauteil, dass je nach Auslenkung den Widerstand im Stromfluss verändern kann, wird nun dieser Stromfluss ausgelesen kann genau bestimmt werden, wie der Potentiometer ausgerichtet ist. Das vorhandene Joystickmodul hat bereits einen internen Schaltkreis, der den eingehenden und ausgehenden Stromfluss mit den eingebauten Potentiometern verbindet.

Zum Auslesen der Joystick Daten mit dem Arduino müssen wir die Pins des Joystick Moduls nutzen. Wir bilden einen Stromkreis mit den 5V Pins des Arduinos mit dem +5V Pin des Joystickmoduls und dem GND-Pin des Arduinos mit dem GND-Pin des Joystickmoduls. Dadurch fließt nun durch das Joystickmodul Strom. Diesen können wir jedoch noch nicht auslesen, dazu müssen wir die VRx und VRy Pins des Joystickmoduls mit beliebigen Analog Pins des Arduino verbinden. Diese erkennt man an der typischen Kennzeichnung mit einem A und dann einer Zahl, der Pin Nummer, die wir später für die Programmierung brauchen, um auch die richtigen Daten auszulesen.

Joystick anschließen

Der Arduino wird auf dem Breadboard angebracht, um das Verbinden mit Strom und dem Joystick zu vereinfachen.

Ein Bild, das Diagramm enthält.

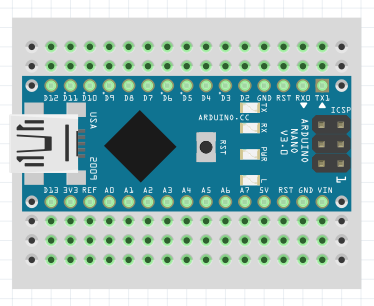
Automatisch generierte Beschreibung

Abbildung 2 Verbindungen auf dem Breadboard. Jede durchgezogene Linie steht für eine Verbindung der darunter liegenden Anschlüssen.

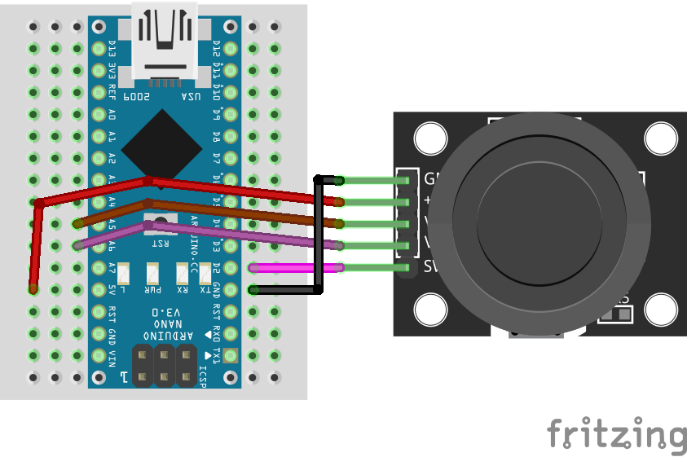
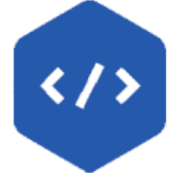
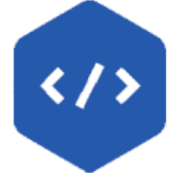
Die Schaltung des Joysticks ist sehr simpel. Da der Joystick als Modul ein eigenes Leiterbahnensystem besitzt muss nur noch die Anschlüsse des Joysticks mit dem Arduino verbunden werden. Dazu nutzen wir die Informationen aus der Einführung, welche Pins das Modul besitzt und wie sie an den Arduino anzuschließen sind.

Abbildung 3 Schaltbild zum Anschluss eines Joystick Moduls an den Arduino Nano

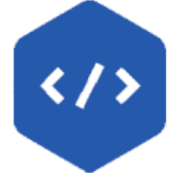
Wenn nun alles an Ort und Stelle ist wollen wir natürlich auch die Daten des Joysticks auslesen. Der Joystick hat drei Daten-Pins: die horizontale Achse (analog), die vertikale Achse (analog) und den Taster (digital). Digitale Pins müssen zu Beginn eines Skripts immer als Eingang oder Ausgang definiert werden. Dazu nutzt man den Befehl pinMode.

Analoge Pins brauchen diese Definition nicht, weshalb wir direkt mit dem Auslesen der Daten anfangen können. Zum Auslesen benutzen wir den Befehl analogRead.

|  |  |
| --- | --- |
| **analogRead(pin);** | |
| **Beschreibung** | Mit diesem Befehl wird der aktuelle Wert an dem angegebenen Pin ausgelesen. |
| **Parameter** | pin: Nummer des Pins, dessen Wert ausgelesen werden soll |
| **Rückgabewert** | Der ausgelesene Wert als Ganzzahl (int) |

Nun könnt ihr die Werte des Joysticks auslesen, wisst jedoch noch nicht, wie man die Werte menschenlesbar anzeigt. Dafür gibt es den seriellen Monitor auf dem man Texte, Werte oder andere nützliche Sachen anzeigen lassen kann. Der Befehl zum Anzeigen von Dingen auf dem seriellen Monitor lautet Serial.print(). Damit die serielle Kommunikation genutzt werden kann, muss in der setup() noch der Befehl Serial.begin() genutzt werden.

|  |  |
| --- | --- |
| **Serial.begin(speed);** | |
| **Beschreibung** | Dieser Befehl wird genutzt, um die serielle Kommunikation mit der angegebenen Geschwindigkeit zu starten. Die Geschwindigkeit ist wichtig, damit |
| **Parameter** | Speed: Die Geschwindigkeit der Kommunikation (Standard ist 9600, später wird auch 38400 benötigt) |
| **Rückgabewert** | void |



|  |  |
| --- | --- |
| **Serial.print(data); / Serial.println(data);** | |
| **Beschreibung** | Dieser Befehl wird genutzt, um nützliche Informationen auf dem seriellen Monitor anzeigen zu lassen. Serial.println() ist eine Erweiterung von Serial.print(), die nach der angezeigten Information automatisch ein Zeilenumbruch macht. |
| **Parameter** | data: Informationen, die auf dem seriellen Monitor angezeigt werden sollen, das kann ein Text sein, eine Variable oder auch Arrays. |
| **Rückgabewert** | Gibt die Anzahl an Zeichen zurück, die tatsächlich geschrieben wurden, falls es zu Problemen mit dem Schreiben kommen sollte. |

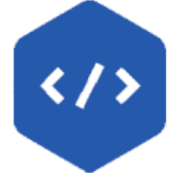
**Aufgabe 1**

|  |  |  |
| --- | --- | --- |
| **1a** | **Schließt den Joystick richtig an den Arduino an und erstellt einen kleinen Sketch, mit dem ihr mit Hilfe von Serial.println() die Werte in dem Debug Fenster ausgebt.** | |
| **1b** | **Was sind die jeweiligen Wertepaare in den äußersten Ecken des Joysticks?**  **Jeweils in der Form (x-Achse, y-Achse).** | |
| Oben Links |  |
| Oben Rechts |  |
| Unten Links |  |
| Unten Rechts |  |

Mit den Daten arbeiten

Infotext (Große Programmieraufgabe, die map Funktion einführen zum mappen von einem Zahlenbereich auf einen anderen und dann mit mit If-Abfragen, interne LED auf Pin 13 erklären und aufblicken lassen, wenn das produkt der beiden Achsen werte kleiner einem bestimmten Wert ist oder so).

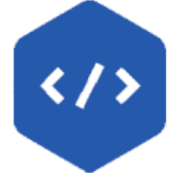
Das Anschließen und Einlesen der Werte haben funktioniert? Dann fangen wir nun an die eingelesenen Werte mal etwas mehr zu benutzen. Die Wertespanne des Joysticks ist, wie sicherlich schon herausgefunden, beschränkt auf Werte von 0 bis 1023. Manchmal ist es jedoch durchaus hilfreich andere Wertespannen zu haben, weil es sich damit leichter arbeiten lässt. Deshalb gibt es den map Befehl.

**Aufgabe 2**

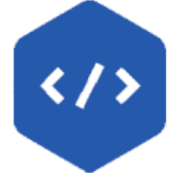
|  |  |
| --- | --- |
| **map(value, fromLow, fromHigh, toLow, toHigh);** | |
| **Beschreibung** | Dieser Befehl arrangiert die Werte so um, dass sie von dem alten Wertebereich nun vollständig in den neuen Wertebereich passen. Dieser Befehl arbeitet nur mit ganzen Zahlen, daher die Grenzen nur als ganze Zahlen angeben. |
| **Parameter** | value: die Variable, dessen Werte neu arrangiert werden sollen  fromLow: die untere Grenze des alten Wertebereichs  fromHigh: die obere Grenze des alten Wertebereichs  toLow: die untere Grenze des neuen Wertebereichs  toHigh: die obere Grenze des neuen Wertebereichs |
| **Rückgabewert** | Die neu arrangierte Ganzzahl mit der dann weitergearbeitet werden kann. |

|  |  |
| --- | --- |
| **2** | **Code schreiben**   * **Lies die x-Werte und y-Werte aus dem Joystick aus und speicher diese in jeweils einer Variable ab** * **Mappe die Werte, die ursprünglich von 0 bis 1023 gingen auf den neuen Wertebereich von -256 bis 255** * **Erstelle eine Bedingung, dass im seriellen Monitor „True“ ausgegeben wird, wenn der x-Wert addiert mit dem y-Wert größer als oder gleich 0 ist. Wenn kleiner, soll „False“ ausgegeben werden.** |

Den Taster auslesen

Nun beschäftigen wir uns mit dem Taster des Joysticks, der erkennt, ob der Joystick gedrückt wurde. Dazu gibt es den digitalen SW-Pin. Bei digitalen Bauteilen/Anschlüssen müssen wir den Arduino darüber informieren, ob über diesen Pin Daten empfangen oder wir Daten ausgeben wollen. Für diese Definition gibt es den pinMode Befehl.

|  |  |
| --- | --- |
| **pinMode(pin, mode);** | |
| **Beschreibung** | Der Befehl legt fest, ob ein bestimmter digitaler Pin als Eingabe oder Ausgabe benutzt werden soll. Wird in void setup() eingesetzt. |
| **Parameter** | pin: Nummer des Pins, dessen Modus gesetzt werden soll  mode: INPUT (Eingabe) oder OUTPUT (Ausgabe), es gibt auch noch den Spezial Modus INPUT\_PULLUP (Eingabe), den wir für den Joystick Taster benötigen werden. |
| **Rückgabewert** | void |

Wenn wir nun auslesen wollen, ob der Joystick gedrückt wurde, müssen wir den Befehl digitalRead nutzen.

|  |  |
| --- | --- |
| **digitalRead(pin);** | |
| **Beschreibung** | Mit diesem Befehl wird an dem angegebenen digitalen Pin ausgelesen, ob der Wert des Pins HIGH oder LOW ist. Der Joystick-Taster gibt immer einen LOW-Wert zurück und wenn gedrückt wechselt er für die Zeit des Drückens auf HIGH |
| **Parameter** | pin: Nummer des Pins, dessen Wert ausgelesen werden soll |
| **Rückgabewert** | HIGH oder LOW |

Das Problem nun ist, dass wir damit nur abfragen können, ob der Taster aktuell gedrückt wird. Wenn gespeichert werden soll, dass der Taster gedrückt wurde, wird es etwas komplizierter. Es muss gespeichert werden, ob der Taster aktuell gedrückt ist (kann auch in der Abfrage direkt ausgelesen werden), es muss gespeichert werden, ob der Taster bei der letzten Abfrage gedrückt war (bool) und es muss einen Modus geben, der hin und her springt, wenn der Taster einmal vollständig gedrückt und losgelassen wurde (enum).

**Aufgabe 3**

|  |  |
| --- | --- |
| **3** | **Code erweitern**   * **Lege eine bool Variable an, die speichert, ob der Taster beim letzten Mal gedrückt war.** * **Lege einen enum an, der zwei verschiedene Modi definiert.** * **Schreibe eine Funktion, die in den jeweils anderen Modus wechselt, wenn die bool Variable auf true ist und der aktuelle Taster LOW ausgelesen wird (nach dem Wechseln nicht vergessen die bool Variable auf false zu setzen). Am Ende der Funktion soll die bool Variable nochmal auf den aktuellen Taster-Wert gesetzt werden.** * **Zu guter Letzt soll die Bedingung aus der Aufgabe 2 so geändert werden, dass wenn der erste Modus aktiviert ist, soll weiterhin „True“ ausgegeben werden, wenn xAchse PLUS yAchse größer oder gleich 0 ist, aber wenn der zweite Modus aktiviert ist, soll „True“ ausgegeben werden, wenn xAchse MAL yAchse größer oder gleich 0 ist.** |