WAVE-O-MAT 2000

Gruppe 05

Studiengang Systemtechnik

Oliver Schmid

Joël Koch

Inhaltsverzeichnis

1. Architektur

RAHMENBEDINGUNGEN

Dies ist eine Abschlussarbeit in Embedded System, welche mithilfe des Roboters „miniQ 2WD“ gelöst werden soll. Der Roboter ist von DFrobot und ist im freien Markt erhältlich. Das System basiert auf einer Arduino Leonardo Plattform. Die Aufgabe besteht darin zwei Programme mit freiwählbaren Funktionen zu Programmieren. Die Ganzen Rahmenbedingungen sind im Dokument <<PA_Aufgabenstellung_mit_Notizen>> zu entnehmen.

IDEE

Der MiniQ verfügt über zwei Helligkeitssensoren, die als Spannungsteiler angeordnet sind. Dadurch kann über den analogen Eingang das Verhältnis der beiden Helligkeiten zueinander erfasst werden. Wird nun Beispielsweise. ein Finger gleichmässig hin- und her bewegt vor den Sensoren, können Grundfrequenz und Amplitude der Bewegung ermittelt werden. Wird eine Frequenz erkannt, wird sie übernommen, wird keine erkannt, so bleibt die bisherige Frequenz stabil. Das ermöglicht es, eine Frequenz am Roboter anzuregen, die er auch ohne Bewegung vor dem Sensor behalten kann.

Während das Programm 1 läuft, leuchten die RGB-LEDs. Die Helligkeit pulsiert dabei mit der Frequenz, die aus der Bewegung ermittelt wurde.

Während das Programm 2 läuft, wird die Frequenz der Bewegung in einen hörbaren Bereich übersetzt und auf dem Lautsprecher als Ton ausgegeben. Die Eingangsfrequenz wird auf ein Band von 1 Hz bis 10 Hz limitiert und anschliessend linear auf einen Frequenzbereich von 100 Hz bis 10 kHz übersetzt.

FUNKTIONSBESCHREIBUNG

In einem Hauptmenu soll mit einem Tastendruck (Key1=Programm1/ Key2=Programm2) das Programm 1 oder 2 gestartet werden. Die Programme laufen dann kontinuierlich ab und zeigen je nach Programm anhand Ton oder RGB-LED die momentane vorgegeben Frequenz an. Auf dem Display soll während dem Programmlauf die aktuelle angezeigte Frequenz sichtbar sein. Mit der Betätigung der Key3 sind die Programme jederzeit zu beenden.

Funktionale Spezifikation für das Gesamtsystem WAVE-O-MAT

	Funktion	Ausgangslage	Ereignis	Vorgang	Ergebnis Soll
1	Menu/Start/Stop	LCD-Display zeigt Auswahlmöglichkeiten an			Anzeige des Menüs
1.1	Start Programm 1	Hauptmenü auf LCD	Impuls an Key1	Programm1 startet	Beep/ Programm1 läuft
1.2	Start Programm 2	Hauptmenü auf LCD	Impuls an Key2	Programm2 startet	Beep/ Programm2 läuft
1.3	Abbruch des aktuellen Programmes	Programm 1 od. 2 läuft	Impuls an Key3	Programm wird beendet und kehrt ins Hauptmenü zurück	Beep/ Programm 1 od. 2 wird beendet
2	Programm 1	Programm 1 gestartet/ RGB-Lampe leuchtet/LCD zeigt Frequenz an			RGB-LED blinkt entsprechend der vorgemachten Frequenz
2.1	Frequenzerkennung	Programm1 läuft	Mit Fingern Frequenz vorgeben	Frequenz wird erkannt und verarbeitet	Frequenz wird ermittelt und weitergegeben
2.2	LCD-Ausgabe	Programm1 läuft		Frequenz wird an LCD-Ausgabe weitergeleitet	Aktueller Wert auf dem LCD angezeigt
2.3	RGB-Ausgabe	Programm1 läuft		Frequenz wird in Helligkeitsfunktion umgerechnet	Aktuelle Frequenz wird blinkend dargestellt
3	Programm 2	Programm 2 gestartet/ Buzzer tönt /LCD zeigt Frequenz an			Buzzer tönt entsprechend der vorgemachten Frequenz
3.1	Frequenzerkennung	Programm2 läuft	Mit Fingern Frequenz vorgeben	Frequenz wird erkannt und verarbeitet	Frequenz wird ermittelt und weitergegeben
3.2	LCD-Ausgabe	Programm2 läuft		Frequenz wird an LCD-Ausgabe weitergeleitet	Aktueller Wert auf dem LCD angezeigt
3.3	Tone-Ausgabe	Programm2 läuft		Frequenz wird in Tonfrequenzen umgerechnet	Aktuelle Frequenz ertönt anhand einer Melodie

2. Design

MODULARISIERUNG

Die Modularisierung hat zum Ziel, das Projekt in möglichst voneinander unabhängige, verschiedene Funktionsmodule aufzuteilen. Der Gesamtarbeitsumfang teilt sich auf die verschiedenen Module auf, wodurch einzelne Module einfacher zu realisieren sind. Durch eine sinnvolle Aufteilung und erstellen effizienter Schnittstellen wird ebenfalls die Übersichtlichkeit der Lösung verbessert. Weiterhin ist es bei einer entsprechend ausgerichteten Modularisierung möglich, einzelne Module vollständig durch Libraries zu ersetzen.

Menügeführte Schnittstelle

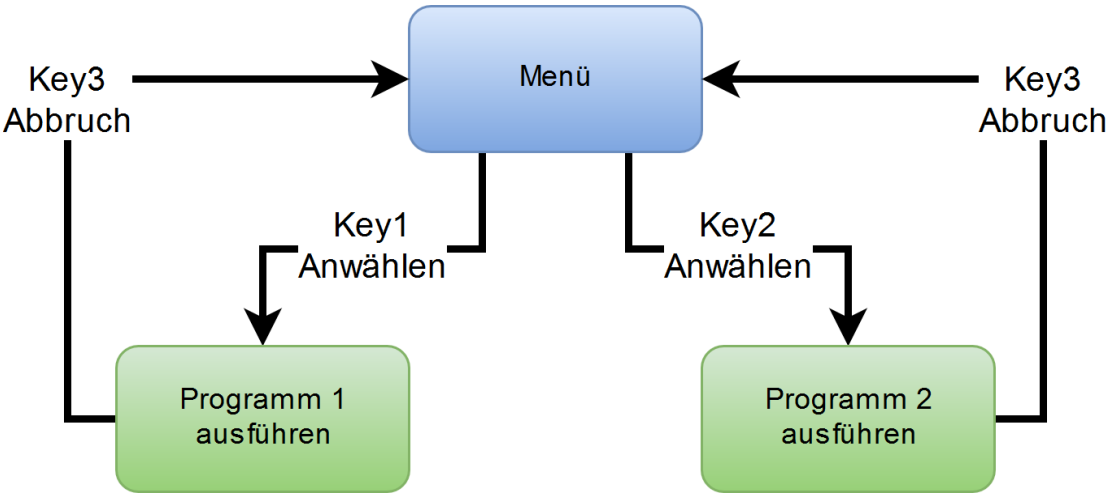

Abbildung 1 Menu

LCD Anzeige

Die LCD Anzeige wird grösstenteils durch die Arduino-Bibliothek «LiquidCrystal_I2C» abgedeckt. Aufgrund der zeitlichen Vorgaben wird pro Interaktion mit dem Displaycontroller jeweils nur ein Zeichen übermittelt oder der Cursor umpositioniert. Das Modul beinhaltet einen eigenen Speicher, in dem der Soll Display Inhalt abgebildet ist. In regelmässigen Abständen wird durch den Scheduler ein Miniupdate gestartet. Pro Miniupdate wird immer nur ein Zeichen an den Controller gesendet. An bestimmten Stellen erfolgt anstelle der Zeichenübermittlung eine Cursorpositionierung, bspw. vor dem ersten Zeichen. Ein Update des gesamten LCD auf einmal würde aufgrund der lange dauernden i2c Busaktivität zwischen MC und Displaycontroller länger als 2 ms dauern. Ein Miniupdate erfolgt in weniger als 2 ms. Bis der gesamte Display Inhalt (2x16 Zeichen) aktualisiert ist, müssen 34 Miniupdates erfolgen.

Soundmodul

Das Soundmodul ermöglicht es, dass kontinuierlich ein Ton am Lautsprecher ausgegeben wird. Als Input wird eine Frequenz im Bereich 1 bis 10 Hz angegeben. Das Modul rechnet die angegebene Frequenz in den hörbaren Bereich um und gibt diesen Ton aus. Die eigentliche Tonerzeugung wird von der Arduino Funktion "tone" übernommen.

KeyRead

Hier wird der analoge Eingang der Taster gelesen und anschliessend der ADC Wert in Integerwerte der entsprechenden Taste übersetzt und zurückgemeldet. Mögliche Rückgabewerte sind 0 (kein Tastendruck), 1, 2 oder 3. Es findet keine Tastenentprellung statt. Durch die Auslegung der Menustruktur sowie der Tatsache, dass in den Programmen keine Tastendrücke ausser die für das Beenden ausgewertet werden müssen, ergibt sich eine vereinfachte Situation: Eine Taste die in einem der Programmzustände (Menu, Programm 1, Programm 2) gedrückt wird und zu einem Zustandswechsel führt, hat im neuen Zustand keine Funktion. Somit ist eine Entprellung nicht notwendig.

Scheduler

Der Scheduler wird über eine ISR periodisch aufgerufen. Der Scheduler besitzt mehrere Jobs. Jeder Job hat einen Divisor und eine Bitmaske. Der Divisor hat die Funktion, den Job nicht jedes Mal zu berücksichtigen, sondern nur jedes n-te Mal. Damit werden auch grössere Intervalle möglich. Wird der Job berücksichtigt, entscheidet das Bit an der Stelle n in der Bitmaske darüber, ob der Job tatsächlich ausgeführt wird oder nicht. Beim nächsten Mal entscheidet dann das Bit an der Stelle n+1 etc. Die Bitmaske ist ein Mittel, um sicherzustellen, dass zeitintensive Jobs nie direkt nach einem anderen zeitintensiven Jobs ausgeführt werden: Indem sie den identischen Divisor haben (oder ein ganzzahliges Mehrfaches des anderen Divisors) und im Bitmuster keine Einsen an den selben Stellen zu finden sind, wird der Job niemals innerhalb des gleichen Scheduler Aufrufs ausgeführt.

Sensor Vorverarbeitung

Die letzten n Werte werden in einem Array gespeichert, da die Datenauswertung den Wertverlauf über eine Zeit benötigt. Zuerst wird das Array um eine Stelle kopiert, so dass der älteste Wert durch den zweitältesten Wert überschrieben wird etc. Der Sensor wird gelesen und der gelesene Wert geglättet und als neuester Wert ins Array geschrieben. Die Glättung erfolgt, indem nur ein Teil der Differenz zwischen dem letzten gespeicherten Wert im Array und dem aktuell gemessenen Wert addiert wird zum zuletzt gespeicherten Wert.

Datenanalyse

Die Datenanalyse benötigt Daten aus einer eher grösseren Periode, bspw. 1 Sekunde. Aus den geglätteten Samples wird die Grundfrequenz und Amplitude (Peak to Peak) bestimmt. Die Amplitude entspricht der Differenz des minimalen zum maximalen

FFT funktionierte nach Tests unzuverlässig. problematisch ist die zu niedrige Frequenz Auflösung bei 30 Hz Samplingrate. Die Samplingrate kann nicht beliebig erhöht werden wegen dem Speicherbedarf der Samples. Als Alternative dazu wurde einen neuen Ansatz mit Flankenerkennung gewählt: im Signal werden die wechselnden Nulldurchgänge und auch die Min und Max Werte (Peaks) erkannt.

1. DC Offset wird rausgerechnet. uint8 auf int8 cast: Werte Bereich muss vorgängig begrenzt werden.

2. Nulldurchgänge werden gesucht. Auf einen steigenden Durchgang muss ein sinkender folgend etc. Die zeitlichen Abstände zwischen den Nulldurchgängen werden gespeichert.

3. Peaks werden gesucht. Auf einen Max. muss ein Min folgen. Die Werte müssen mindestens den Hysterese-Abstand zwischen einander haben. Die zeitlichen Abstände werden gespeichert.

4. Die zeitlichen Abstände werden sortiert.

5. Eine Art Median der Abstände wird ermittelt. Mittelwert der 3 zentralen Werte, wobei der mittlere doppelt gewichtet wird.

6. Umrechnen in eine Frequenz "rawfrequency" (ist aktuell die Periodendauer einer Halbwelle in Anzahl Samples)

7. Plausibilitätscheck, ansonsten wird die zuletzt als gültig erkannte Frequenz wiederverwendet.

8. Es erfolgt eine Lowpassfilterung der Frequenz.

LED-update

Die RGB-LED wird mit Hilfe der Adafruit Neopixel-Library angesteuert. Wir gehen von einem Dreiecksinus der gegebenen Frequenz aus. In einem ersten Schritt wird davon die Steigung berechnet. Danach mit der Aufrufrate die Intensität berechnet, welche anschliessend auf die RGB-LED ausgegeben wird. Da dieser Anstieg sehr hart ist und das menschliche Auge dies als störend empfindet, wird das Signal mit einer Gammafunktion dem Auge angepasst.

MODULHIERARCHIE

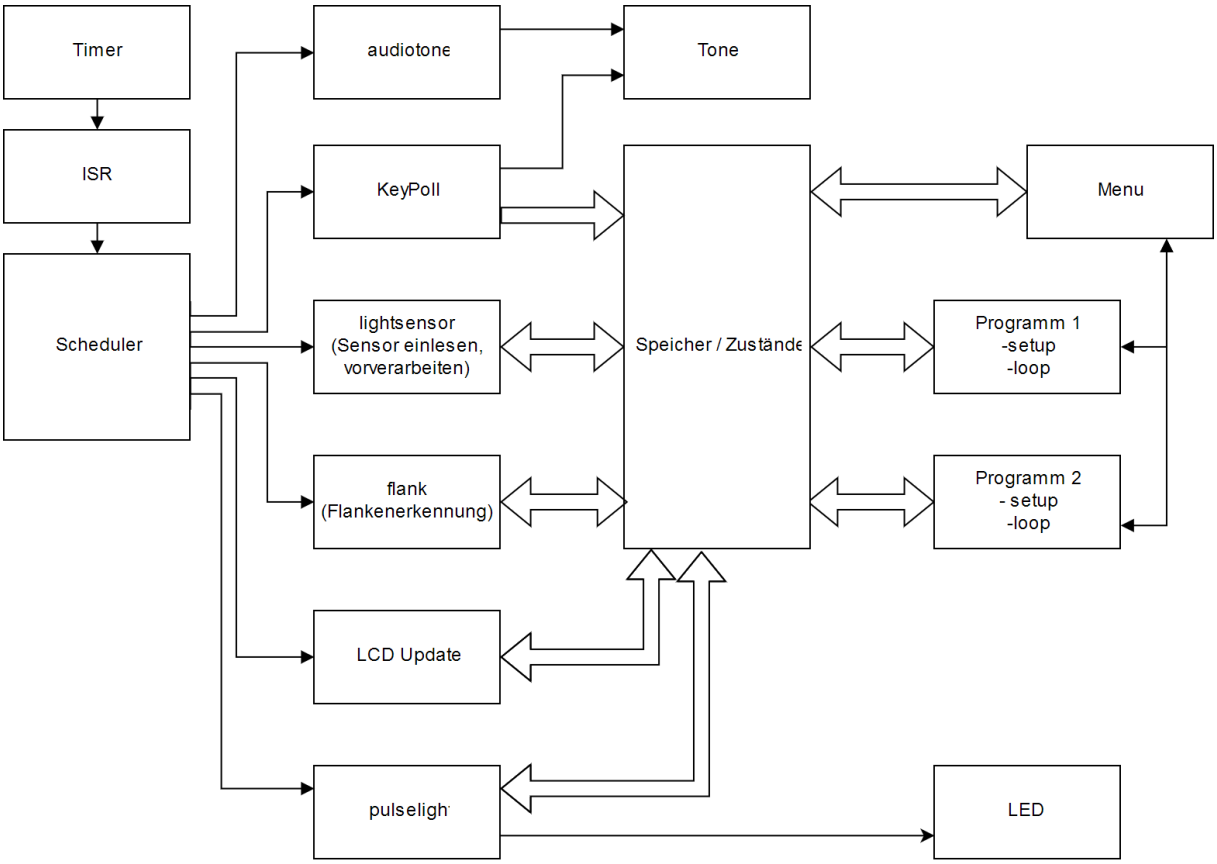

Abbildung 2 Modulhierarchie

SCHNITTSTELLENDEFINITIONEN

Modul Job

Job();	Initialisierung
bool now(int cycleCounter);	
void stop();	Startet Job
void start();	Stoppt Job
void init(int cycleConfigPattern, int divider,void (*exe)());	Initialisierung
void (*exe)();	

Modul JobController

JobController();	
void check();	Wird in regelmässigen Abständen aufgerufen.
void stopjob(int jobnr);	Einzelne Jobs können gezielt gestoppt werden
void startjob(int jobnr);	Einzelne Jobs können gezielt gestartet werden
void enable();	Der JobController als Ganzes wird aktiviert.
void disable();	Der JobController als Ganzes wird dektiviert.
void init(int jobnr, int cycleConfigPattern, int divider, void (*exe)());	Initialisierung

Modul Lightsensor

Lightsensor();	Erstellt Buffer
void update();	Erfasst neuen Wert und Filterung
uint8_t data[LBUFFERSIZE];	Globale Variable

Modul Audiotone

Audiotone(void);	Initialisierung
~Audiotone(void);	Destruktor mit Abschalten des Buzzers
void update(double lowfreq);	Aktualisiert die ausgegebene Frequenz an den Buzzer
void stop(void);	Abschalten des Buzzers

Modul Flank

Flank(double samplefrequency);	Initialisierung
double frequency(void);	Gibt verarbeitete Frequenz aus
double frequencyraw(void);	Gibt unverarbeitete Frequenz aus
void calculate(int8_t * vdat);	Berechnet Frequenz
void printArray(int8_t * v, char * title, uint8_t len);	Array auf die Serielle Schnittstelle ausgeben
void printTn();	Die ermittelten Intervalle (Nulldurchgang/Nulldurchgang und Peak/Peak)

Modul Keypoll

int keypoll();	Wertet die Tasten aus. Rückgabewert 1-3 für Tasten und 0 für keine betätigt.

Modul pulselight

void LED_setup();	Initialisierung
void LED_end();	Schaltet RGB-LED aus
double LED_setfq(double fq);	Rechnet die Frequenz in eine Steigung um und gibt diese zurück
void LED_update(double m);	Berechnet die Helligkeit und zeigt diese an

Modul update_LCD

void update_LCD(char * eingabe);	Setzt Cursor/ gibt das Chararray einzeln aufs LCD aus
void LCD_setup();	LCD-Initialisierung
void LCD_clear();	Löscht das LCD

3. Test

FUNKTIONALE SPEZIFIKATION

Die Funktionalen Spezifikationen dienen idealerweise als Test-Anweisungen für den Modul-, Integrations- und Abnahmetest. Die Zeitkritischen Funktionen werden mit Hilfe der Funktion millis() auf Laufzeit getestet, um sicherzustellen, dass die 2ms Grenze nicht überschritten wird.

Modul	Methode	Ausgangslage	Ereignis	Vorgang	Ergebnis Soll	Ergebnis Ist
pulselight	LED_setfq	Frequenzen vorgegeben	LED_setfq aufrufen	Kein Beobachter	Frequenz in Steigung umrechnen	Werte plausibel
pulselight	LED_update	Setfq ausgeführt	LED_update zyklisch aufrufen	RGB_LED ändert Intensität	Licht pulsiert in der Frequenz	-LED-pulsiert gemäss vorgegebener Frequenz -Ein Durchgang 0.6ms
Pulselight	LED_end	LED leuchtet	LED_end	LED stellt ab		Die LED geht aus
Audiotone	update	Frequenzen vorgegeben	Update zyklisch mit verschiedenen Frequenzen aufrufen	Ton ertönt		Ton ertönt in hörbarer Höhe
Audiotone	stop	Buzzer läuft	Stop aufrufen	Ton stellt ab		Ton stellt ab
Update LCD	LCD_update	Chararray 34 Zeichen	LCD_update zyklisch aufrufen	Setzt Cursor und Zeichen einzeln	Text erscheint auf dem Display	Die Buchstaben erscheinen nacheinander 32Zeichen +2 Cursor setzen=53ms 53ms/34→1.6ms
flank	update		Finger vor den Sensoren hin und her	Frequenz wird ermittelt	Die Frequenz sollte mit der vorgegebenen Frequenz der Finger übereinstimmen	Die Frequenz wird nicht immer so sauber erkannt. Zum einen, weil sich die Sensorhardware nicht so gut für das Vorhaben eignet. Zum andern besteht beim flank.cpp noch Verbesserungspotential, insbesondere bei der Selektion, welche Frequenzen richtig erkannt wurden und welche verworfen werden müssen.
Lightsensor	Update		Wert einlesen, LowpassfilterN		Array geschoben. Neuer Wert ist geglättet.	Array geschoben, Neuer Wert ist geglättet.

TESTSPEZIFIKATION GESAMTSYSTEM WAVE-O-MAT

	Funktion	Ausgangslage	Ereignis	Vorgang	Ergebnis Soll	Ergebnis Ist
1	Menu/Start/Stop	LCD-Display zeigt Auswahlmöglichkeiten an			Anzeige des Menus	Menu wird auf LCD angezeigt
1.1	Start Programm 1	Hauptmenü auf LCD	Impuls an Key1	Programm1 startet	Beep/ Programm1 läuft	Beep ertönt Programm1 läuft
1.2	Start Programm 2	Hauptmenü auf LCD	Impuls an Key2	Programm2 startet	Beep/ Programm2 läuft	Beep ertönt Programm2 läuft
1.3	Abbruch des aktuellen Programmes	Programm 1 od. 2 läuft	Impuls an Key3	Programm wird beendet und kehrt ins Hauptmenü zurück	Beep/ Programm 1 od. 2 wird beendet	Beep ertönt/ Hauptmenü erscheint/ Ton od. LED geht aus
2	Programm 1	Programm 1 gestartet/ RGB-Lampe leuchtet/LCD zeigt Frequenz an			RGB-LED blinkt entsprechend der vorgemachten Frequenz	LED pulsiert gemäss der angezeigten Frequenz
2.1	Frequenzerkennung	Programm1 läuft	Mit Fingern Frequenz vorgeben	Frequenz wird erkannt und verarbeitet	Frequenz wird ermittelt und weitergegeben	Angezeigte Frequenz ist Plausibel
2.2	LCD-Ausgabe	Programm1 läuft		Frequenz wird an LCD-Ausgabe weitergeleitet	Aktueller Wert auf dem LCD angezeigt	Frequenz wird auf dem LCD aktualisiert
2.3	RGB-Ausgabe	Programm1 läuft		Frequenz wird in Helligkeitsfunktion umgerechnet	Aktuelle Frequenz wird blinkend dargestellt	LED pulsiert
3	Programm 2	Programm 2 gestartet/ Buzzer tönt /LCD zeigt Frequenz an			Buzzer tönt entsprechend der vorgemachten Frequenz	Ton ertönt gemäss der angezeigten Frequenz
3.1	Frequenzerkennung	Programm2 läuft	Mit Fingern Frequenz vorgeben	Frequenz wird erkannt und verarbeitet	Frequenz wird ermittelt und weitergegeben	Angezeigte Frequenz ist Plausibel
3.2	LCD-Ausgabe	Programm2 läuft		Frequenz wird an LCD-Ausgabe weitergeleitet	Aktueller Wert auf dem LCD angezeigt	Frequenz wird auf dem LCD aktualisiert
3.3	Tone-Ausgabe	Programm2 läuft		Frequenz wird in Tonfrequenzen umgerechnet	Aktuelle Frequenz ertönt anhand einer Melodie	Ton entsprechend der Frequenz ertönt