Maestría en Sistemas Inteligentes Multimedia

Compiladores y desarrollo de librerías

Proyecto Final: Desarrollo de librería para motor a pasos

Héctor Edmundo Ramírez Gómez

Junio 24 2017

Contenido

Introducción

Motor a pasos

Los motores a pasos convierten impulsos eléctricos en sus terminales con cierta secuencia en movimientos angulares discretos. Debido a esta característica los motores a pasos pueden ser gobernados fácilmente mediante sistemas digitales como microcontroladores.

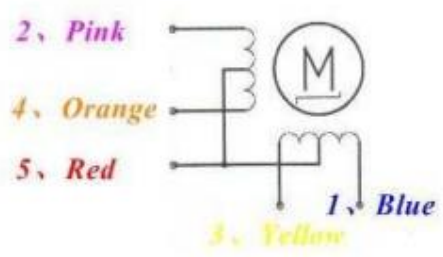
Motor a pasos 28BYJ-48
Relación de engranes 64:1
Unipolar, no requiere cambios de polaridad para avanzar

Driver ULN2003 (Arreglo Darlington de Transistores)
Permite acoplar la baja corriente y bajo voltaje del microcontrolador a las altas demandas del motor a pasos

Modos de operación

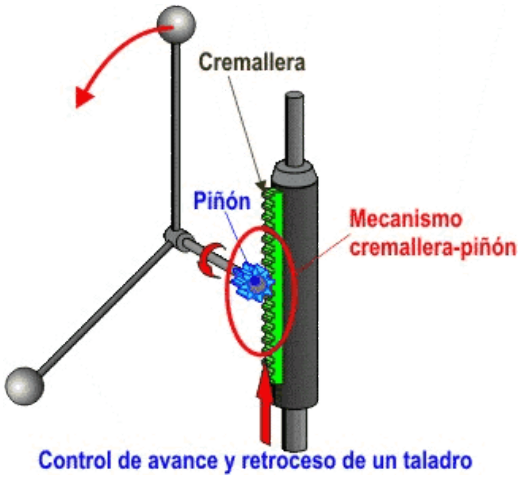
Se emplea el modo de operación de paso completo (dos fases por paso) ya que brinda un mayor torque, necesario para la aplicación final.

METHODS	PHASES	STEPS 1	2	3	4	5	6	7	8
WAVE DRIVE One phase at a time Simplest, but least used	BLUE								
	PINK								
	YELLOW								
	ORANGE								
FULL STEP Two phases at a time Strongest Torque	BLUE								
	PINK								
	YELLOW								
	ORANGE								
HALF STEP One, or two phases at a time Smallest step angle Medium torque	BLUE								
	PINK								
	YELLOW								
	ORANGE								



Piñón y cremallera

Un mecanismo de cremallera es un dispositivo mecánico con dos engranajes, denominados «piñón» y «cremallera», que convierte un movimiento de rotación en un movimiento rectilíneo o viceversa. El engranaje circular denominado «piñón» engrana con una barra dentada denominada «cremallera», de forma que un giro aplicado al piñón causa el desplazamiento lineal de la cremallera



Control de avance y retroceso de un taladro

Plataforma

Raspberry Pi 3B

Raspberry Pi es una computadora de placa reducida, computadora de placa única o computador de placa simple (SBC) de bajo costo desarrollada en Reino Unido por la Fundación Raspberry Pi. La Raspberry Pi usa mayoritariamente sistemas operativos GNU/Linux. Raspbian, una distribución derivada de Debian que está optimizada para el hardware de Raspberry Pi.

Esta plataforma brinda flexibilidad en el uso de herramientas de software, y para el caso de este proyecto se usó el lenguaje de programación C++ con el compilador g++, dirigido por un makefile.

Desarrollo de código y librería

Para la implementación del código se usó la librería pigpio, que permite un acceso fácil a los puertos de entrada y salida de la raspberry. Además, se desarrolló una propia librería para controlar el motor a pasos.

Modos de operación

El proyecto cuenta con dos modos de operación; manual y automático. También cuenta con dos entradas digitales que controlan el avance hacia adelante y hacia atrás. Cuando el sistema se encuentra en modo manual, el motor avanza indefinidamente en el sentido cuya entrada se active. En modo automático, las entradas son sólo pulsos y el accionamiento del motor es por la fracción de vuelta que se configure.

Código fuente

Función principal

```
#include <pigpio.h>
#include <unistd.h>
#include <stdio.h>
#include "../libraries/direction.h"


int main(){
    Direction dir;
    //GPIO Initialize
    if (gpioInitialise()<0) return -1;

    //Stepper motor gpio configurartion
    gpioSetMode(dir.getA(),PI_OUTPUT);
    gpioSetMode(dir.getB(),PI_OUTPUT);
    gpioSetMode(dir.getC(),PI_OUTPUT);
    gpioSetMode(dir.getD(),PI_OUTPUT);

    //Buttons configs
    int forward = 02;
    int reverse = 03;
    int manual = 04;
    int active = 17;


    //Control buttons gpio configuration
```

```
    gpioSetMode(forward, PI_INPUT);
    gpioSetMode(reverse, PI_INPUT);
    gpioSetMode(manual, PI_INPUT);
    gpioSetMode(active, PI_INPUT);

    while(gpioRead(active)){
        if(gpioRead(forward) && !gpioRead(manual)){
            dir.forward(0.25);
        }
        if(gpioRead(forward) && gpioRead(manual)){
            dir.forwardManual();
        }
        if(gpioRead(reverse) && !gpioRead(manual)){
            dir.reverse(0.5d);
        }
        if(gpioRead(reverse) && gpioRead(manual)){
            dir.reverseManual();
        }
    }

    dir.stop_motor();
    gpioTerminate();
    return 0;
}
```

Header File

```
#ifndef DIRECTION_H
#define DIRECTION_H


class Direction{

      int A;
      int B;
      int C;
      int D;
      int delay_us;

      public:
      Direction();
      void forward(float c);
      void forwardManual();
      void reverse(float c);
      void reverseManual();
      void stop_motor();

      int getA();
      int getB();
      int getC();
      int getD();
      int getDelay_us();

      void setA(int x);
      void setB(int x);
      void setC(int x);
      void setD(int x);
      void setDelay_us(int x);
};

#endif
```

Librería

```
#include "../libraries/direction.h"
#include <pigpio.h>
#include <unistd.h>
#include <stdio.h>

Direction::Direction(){
    A = 12;
    B = 16;
    C = 20;
    D = 21;
    delay_us = 3000;
}

void Direction::forward(float c){
    printf("Automatic Forward\n");
    float i = c * 256;
    for(float x; x<i;x++){
        //Step 1
        gpioWrite(A,1);
        gpioWrite(B,1);
        gpioWrite(C,0);
        gpioWrite(D,0);
        usleep(delay_us);

        //Step 2
        gpioWrite(A,0);
        gpioWrite(B,1);
        gpioWrite(C,1);
        gpioWrite(D,0);
        usleep(delay_us);

        //Step 3
        gpioWrite(A,0);
        gpioWrite(B,0);
        gpioWrite(C,1);
        gpioWrite(D,1);
        usleep(delay_us);

        //Step 4
        gpioWrite(A,1);
        gpioWrite(B,0);
        gpioWrite(C,0);
        gpioWrite(D,1);
        usleep(delay_us);

        //Step 5
        gpioWrite(A,1);
        gpioWrite(B,1);
```

```
        gpioWrite(C,0);
        gpioWrite(D,0);
        usleep(delay_us);

        //Step 6
        gpioWrite(A,0);
        gpioWrite(B,1);
        gpioWrite(C,1);
        gpioWrite(D,0);
        usleep(delay_us);

        //Step 7
        gpioWrite(A,0);
        gpioWrite(B,0);
        gpioWrite(C,1);
        gpioWrite(D,1);
        usleep(delay_us);

        //Step 8
        gpioWrite(A,1);
        gpioWrite(B,0);
        gpioWrite(C,0);
        gpioWrite(D,1);
        usleep(delay_us);
    }
}

void Direction::forwardManual(){
    printf("Manual Forward\n");
      .
      .
      .
    }

void Direction::reverse(float c){
    printf("Reverse\n");
    float i = c * 256;
    for(float x; x<i;x++){
      .
      .
      .
    }
}

void Direction::reverseManual(){
    printf("Reverse Manual\n");
        //Step 8
        gpioWrite(A,1);
        gpioWrite(B,0);
        gpioWrite(C,0);
        gpioWrite(D,1);
        usleep(delay_us);
```

```
        //Step 7
        gpioWrite(A,0);
        gpioWrite(B,0);
        gpioWrite(C,1);
        gpioWrite(D,1);
        usleep(delay_us);

        //Step 6
        gpioWrite(A,0);
        gpioWrite(B,1);
        gpioWrite(C,1);
        gpioWrite(D,0);
        usleep(delay_us);

        //Step 5
        gpioWrite(A,1);
        gpioWrite(B,1);
        gpioWrite(C,0);
        gpioWrite(D,0);
        usleep(delay_us);

        //Step 4
        gpioWrite(A,1);
        gpioWrite(B,0);
        gpioWrite(C,0);
        gpioWrite(D,1);
        usleep(delay_us);

        //Step 3
        gpioWrite(A,0);
        gpioWrite(B,0);
        gpioWrite(C,1);
        gpioWrite(D,1);
        usleep(delay_us);

        //Step 2
        gpioWrite(A,0);
        gpioWrite(B,1);
        gpioWrite(C,1);
        gpioWrite(D,0);
        usleep(delay_us);

        //Step 1
        gpioWrite(A,1);
        gpioWrite(B,1);
        gpioWrite(C,0);
        gpioWrite(D,0);
        usleep(delay_us);
}

void Direction::stop_motor(){
```

```
    gpioWrite(A,0);
    gpioWrite(B,0);
    gpioWrite(C,0);
    gpioWrite(D,0);
}

int Direction::getA(){
    return A;
}
int Direction::getB(){
    return B;
}
int Direction::getC(){
    return C;
}
int Direction::getD(){
    return D;
}
int Direction::getDelay_us(){
    return delay_us;
}

void Direction::setA(int x){
    A = x;
}

void Direction::setB(int x){
    B = x;
}

void Direction::setC(int x){
    C = x;
}

void Direction::setD(int x){
    D = x;
}

void Direction::setDelay_us(int x){
    delay_us = x;
}
```

Proceso de compilación

Para el proceso de compilación se empleó la metodología de makefiles, que facilita el proceso en tiempo de desarrollo ya que sólo requiere una simple instrucción.

Makefile

```
build: pre compile asm linker

pre:
	cpp source/stepper.cpp > build/stepper.i
	cpp source/direction.cpp > build/direction.i
compile:
	g++ -S -o build/stepper.s build/stepper.i
	g++ -S -o build/direction.s build/direction.i
asm:
	as -o build/stepper.o build/stepper.s
	as -o build/direction.o build/direction.s
linker:
	g++ -o build/stepper.exe build/stepper.o build/direction.o -
lpigpio -pthread
run:
	sudo build/stepper.exe
clean:
	rm build/stepper.i build/stepper.o build/stepper.s
	rm build/direction.*
```

Implementaciones futuras

- La plataforma elegida es cara y de excedidas capacidades para la aplicación, por lo que se migrará a un microcontrolador más barato y compacto, como Arduino micro, tarjeta Teensy o Raspberry Zero.

- Se requiere una interfaz con el usuario, por lo que se implementará una pantalla LCD y teclado numérico

- Se implementarán botones industriales tipo pulsador y selector para soportar la carga de trabajo.

Código en GitHub

Todo el código de este proyecto está disponible en: https://github.com/edmundormz/stepper.git