Санкт-Петербургский политехнический университет Петра Великого

Курсовая работа

по дисциплине:

«Объектно-ориентированное программирование»

на тему:

«Настройка и доработка программного обеспечения управления трёхосевыми станками с ЧПУ GRBL»

Студент: Табунщик А. М.

Преподаватель: Ананьевский М.С.

Группа: 3331506/10101

Санкт-Петербург

2025

Содержание

Оглавление

Введение 3
1. Модификация прошивки GRBL 4
2. Настройка станка 6
   2.1. Прошивка контроллера 6
   2.2. Установка управляющей программы 6
   2.3. Настройка параметров движения 7
3. Формирование G-code 7
   3.1. Общие сведения о генераторе G-code 7
   3.2. Алгоритм генерации G-code 8
Заключение 9
Список использованных источников 10
Приложения 11

Введение

При производстве инфракрасных (ИК) датчиков на этапе сборки требуется наносить силиконовое покрытие на корпуса и устанавливать линзы. Корпуса датчиков размещаются в специальной матрице, которая затем фиксируется на рабочей поверхности трехосевого станка с ЧПУ, оснащенного дозатором силикона.

Приводы осей X, Y, Z реализованы с использованием ременной передачи и шаговых двигателей NEMA17 с углом шага 1,8°. Дозатор представляет собой шприц с пневматической системой подачи воздуха.

Управление станком основано на выполнении заранее разработанного алгоритма движения. Для минимизации затрат на разработку программного обеспечения было решено адаптировать существующее решение с небольшими доработками. Выбор пал на прошивку GRBL, которая поддерживает выполнение команд в формате G-кода. Программа управления формируется с помощью готового конфигуратора. Для относительного позиционирования на концах осей установлены концевые выключатели, так как шаговые двигатели не имеют встроенной обратной связи по положению.

В качестве управляющей платы используется разработанный контроллер на базе микроконтроллера ATmega328P. Он поддерживает одновременное управление тремя шаговыми двигателями, считывание сигналов с трех концевых выключателей, управление тремя силовыми реле и связь с компьютером через порт USB 2.0.

GRBL выполняет команды, поступающие по USB, поэтому для взаимодействия с пользователем необходим интерфейс верхнего уровня, который может быть реализован на ПК под управлением Windows или Linux. В качестве интерфейса была выбрана программа Candle.

1. Модификация прошивки GRBL

Прошивка GRBL состоит из нескольких библиотек и файлов, каждый из которых отвечает за определенные функции. Например:

- Stepper – формирование импульсов для управления шаговыми двигателями;
- Serial – обеспечение связи через USB;
- Report – обнаружение и сообщение об ошибках в работе устройства;
- Grbl – основная часть прошивки, отвечающая за интерпретацию G-кода, запрос данных с ПК после выполнения команды и т.д.

Настройки портов ввода-вывода можно изменить, отредактировав файл cpu_map.h.

Основные параметры прошивки задаются в файле config.h, но некоторые из них можно изменить через командную строку. Для корректной работы устройства необходимо обеспечить независимое управление тремя реле. Плата была спроектирована так, чтобы изменения в cpu_map.h не требовались.

В стандартной конфигурации GRBL управление шпинделем или лазером осуществляется через сигнал ШИМ на выводе D11, связанном с D13. На разработанной плате три силовых реле подключены к портам D11, D13 и PC3 микроконтроллера ATmega328P. Порт PC3 изначально предназначен для управления насосом подачи СОЖ и имеет независимое управление.

Для обеспечения независимой работы реле в файле config.h необходимо раскомментировать строку

#define USE_SPINDLE_DIR_AS_ENABLE_PIN,

что позволяет порту D13 работать отдельно от D11.

В файле spindle_control.c изменяется вид функции spindle_stop(), комментируется строка:

SPINDLE_TCCRA_REGISTER &= ~(1<<SPINDLE_COMB_BIT);

Таким образом мы отключаем генерацию ШИМ, контакт шпинделя при подаче сигнала будут иметь либо высокий, либо низкий уровень сигнала.

Важный момент, изменения прошивки вступают в силу при использовании режима лазера.

В файле gcode.c блок [4. Set spindle speed] после строки

spindle_sync(gc_state.modal.spindle, 0.0)

добавлена:

spindle_sync(gc_state.modal.spindle, gc_block.values.s);

Это позволяет управлять в режиме лазера функцией включения, выключения с помощью команды “S”, передавая с ней заранее установленное значение.

В файле spindle_control.c в функции spindle_set_state комментируются строчки:

```
#ifdef VARIABLE_SPINDLE
   sys.spindle_speed = 0.0;
 #endif
```

Добавляем строку:

```
spindle_set_speed(spindle_compute_pwm_value(rpm));
```

Закомментированы строки

```
if (settings.flags & BITFLAG_LASER_MODE) {
if (state == SPINDLE_ENABLE_CCW) { rpm = 0.0; }
}
```

Эти изменения отключают автоматическое обнуление значений шпинделя в любом из режимов, оставляя управление полностью за программой пользователя.

Для сборки изменённого ПО разработчиками написан Makefile, также, сборка и компиляция возможны с использованием среды разработки ArduinoIDE с добавлением архива в качестве библиотеки.

2. Настройка станка

2.1. Прошивка контроллера

Плата управления изначально не имеет загрузчика для прошивки через UART. Согласно документации, для прошивки используется интерфейс SPI и программатор USBasp. В среде Arduino IDE процесс выполняется следующим образом:

Скетч -> Подключить библиотеку -> grbl.zip

Открывается main файл, с которого начинается компиляции в выбранной среде разработки:

Файл -> Примеры -> grbl -> grblUpload

Контроллер atmega328p используется в платах ArduinoUNO, имеет характерно для этой платы спроектированную электрическую принципиальную схему, поэтому без изменений в качестве прошиваемой платы необходимо выбрать:

Инструменты -> Плата: -> Arduino/GenuinoUNO

Программатор USB ASP подключается к портам ввода вывода 13, 12, 11, 10, которые имеют функционал SCK, MISO, MOSI, SS соответственно. В прошивке GRBL используется библиотека serial, обеспечивающая возможность использования на плате порта USB, общающегося с контроллером через USB-TTL преобразователь, подключенный к UART0.

Выбор программатора:

Инструменты -> Программатор: -> USBasp

Перед полноценной прошивкой необходимо записать загрузчик:

Инструменты -> Записать загрузчик

Далее загрузка программы:

Скетч -> Загрузка через программатор

2.2. Установка управляющей программы

В качестве программы исполнителя уже подготовленного G-code используется интерфейс Candle. Сборка самой последней версии требует использования QT 5.4.2 с использованием компилятора MinGW/GCC.

В случае с установкой компилятора взята готовая сборка с сайта https://github.com/niXman/mingw-builds-binaries/releases, после чего распакована в необходимый каталог и добавлена в переменную PATH командой:

set PATH=%PATH%;C:\необходимый_путь

2.3. Настройка параметров движения

Параметры движения редактируются через командную строку. Команда $$ отображает текущие настройки. Изменяемые параметры:

$0=5 — минимальная длительность импульса (в микросекундах) для срабатывания драйверов TB6600 (по документации — 3,5 мкс).

$1=255 —задержка отключения двигателей (в миллисекундах). Значение 255 обеспечивает постоянное питание двигателей.

$20=1 — активация программных границ для предотвращения выхода за допустимую область. При нарушении границ GRBL останавливает движение, шпиндель и охлаждение, выдавая сигнал тревоги.

$22=1 — активация цикла поиска начальной позиции для определения базовой точки.

$100-$102=66.667 — количество шагов на 1 мм перемещения, рассчитанное по формуле:$T_{лп} = \frac{S_{шд} \cdot F_{шд}}{P_р \cdot N_{шк}} = \frac{200 \cdot 8}{2 \cdot 12} = 66\frac{2}{3}$, где

Тлп - точность линейного перемещения, шаг/мм

Sшд — количество шагов на оборот для двигателя

Fшд — микрошаг

Pp — шаг ремня

Nшк — количество зубьев на шкиве.

Конфигурации всех трёх ременных передач одинаковы, следовательно значения параметров равны.

3. Формирование G-code

3.1. Общие сведения о генераторе G-code

Генератор написан на языке программирования Python и состоит из библиотеки с описанными функциями инициализации, настройки, а также генерации g-code отдельных объектов и операций и исполняемого файла, в котором описывается порядок работы станка и задаются параметры настроек и инициализации.

Используемые при генерации G коды:

G21 – Устанавливает единицы измерения в миллиметры.

G28 – Возвращает станок в начальное положение.

G92 – Устанавливает текущую позицию как нулевую для указанных осей.

G0 – Быстрое перемещение инструмента.

G1 – Линейная интерполяция (перемещение с заданной скоростью).

G4 – Пауза на заданное время.

M3 – Включение подачи клея (или другого инструмента).

M5 – Остановка подачи клея (или другого инструмента).

3.2. Алгоритм генерации G-code

Инициализация и настройка:

- Установить единицы измерения (например, миллиметры) с помощью команды G21.
- Возврат станка в начальное положение с помощью команды G28, если это необходимо.
- Установить нулевые координаты для осей с помощью команды G92.

Настройка параметров:

- Определить параметры задачи, такие как размеры и расположение объектов, скорости перемещения и т.д.

Основной цикл генерации:

Для каждого объекта или операции:

- Переместить инструмент в начальную позицию с помощью команды G0 (быстрое перемещение).
- Установить необходимую скорость подачи и перемещаться к начальной точке обработки с помощью команды G1.
- Выполнить необходимые операции, такие как рисование прямоугольников, кругов и т.д., используя соответствующие команды.
- Применить дополнительные функции, такие как подача клея или пауза, если это необходимо.

Завершение программы:

- Остановить подачу клея или других инструментов с помощью команды M5.
- Вернуть инструмент в безопасное положение или начальное положение.
- Сохранить сгенерированный G-код в файл для последующего использования на станке с ЧПУ.

Сохранение и вывод:

- Сохранить сгенерированный G-код в файл с расширением .gcode.
- Вывести файл для использования на станке с ЧПУ.

Код программы приведен в приложениях.

Заключение

В рамках курсовой работы была выполнена адаптация и настройка программного обеспечения для управления трехосевым станком с ЧПУ на базе прошивки GRBL. Основные задачи включали модификацию прошивки под требования оборудования, настройку параметров движения и создание G-кода для выполнения операций.

Модификация GRBL обеспечила независимое управление тремя реле, что позволило эффективно управлять дозатором и другими устройствами. Оптимизация параметров повысила точность и надежность работы станка. Настройка движения и генерация G-кода обеспечили выполнение задачи нанесения силиконового покрытия. Интерфейс Candle упростил взаимодействие с оборудованием.

Работа показала возможность адаптации готовых решений для специализированных задач, а результаты могут быть использованы для дальнейшей автоматизации производства ИК-датчиков.

Список использованных источников

1. AT Machining. G-коды для ЧПУ. — URL: https://at-machining.com/ru/g-code-cnc/ (дата обращения: 01.05.2025).
2. CNC-tex. Расчет и настройка ременной и винтовой передачи ЧПУ. — URL: https://cnc-tex.ru/news/35/raschet-i-nastroika-remennoi-i-vintovoipridachi-chpu.html (дата обращения: 02.05.2025).
3. GitHub. MinGW Builds Binaries. — URL: https://github.com/niXman/mingw-builds-binaries/releases (дата обращения: 03.05.2025).
4. Назаров, А. А. Фрезерный станок с ЧПУ на основе открытого программного обеспечения / А. А. Назаров. — Молодой ученый, 2018. — № 22 (208). — С. 166–169. — URL: https://moluch.ru/archive/208/50976/ (дата обращения: 5.05.2025).
5. Прошивка GRBL – настройка параметров на русском языке. — URL: https://cnc-design.ru (дата обращения: 02.05.2025).
6. Как настроить GRBL и управлять станком с ЧПУ на Arduino. — URL: https://cnc-maniac.ru (дата обращения: 02.05.2025).
7. Подробная инструкция по настройке grbl controller: шаг за шагом. — URL: https://zvenst.ru (дата обращения: 01.05.2025).

Приложения

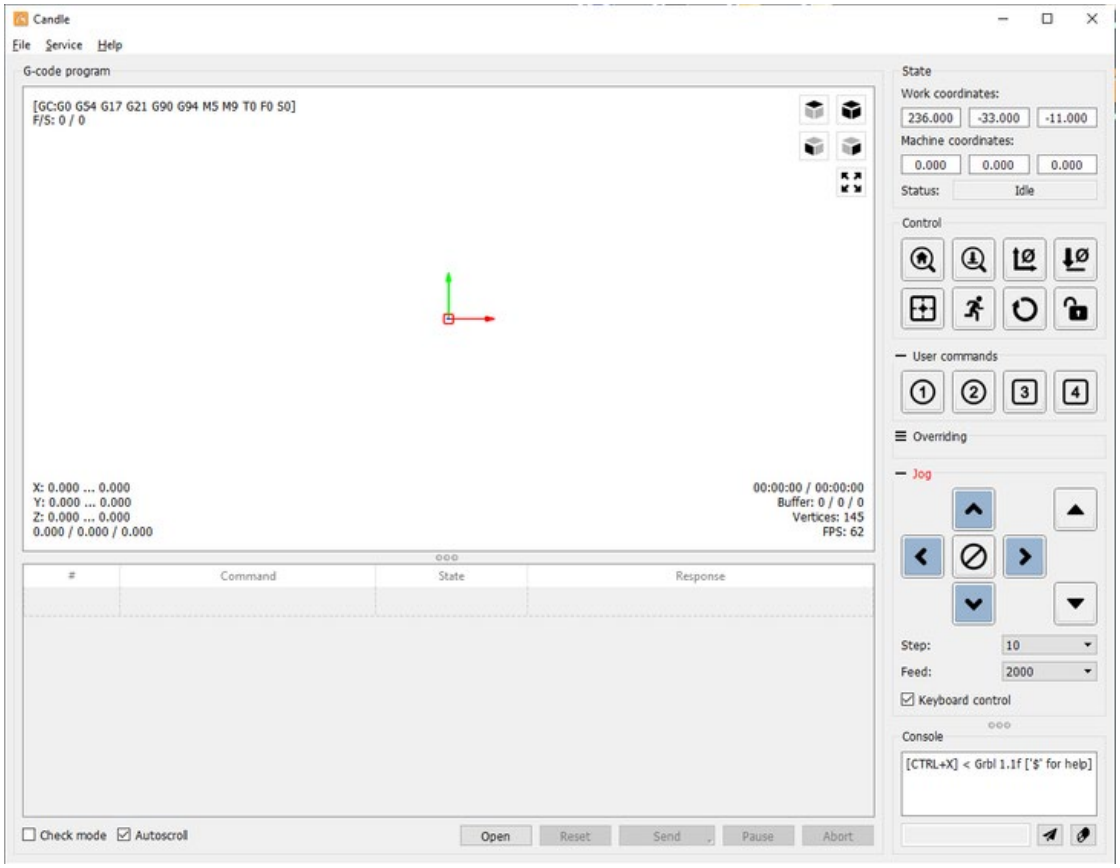

Рисунок 1 – Интерфейс программы candle

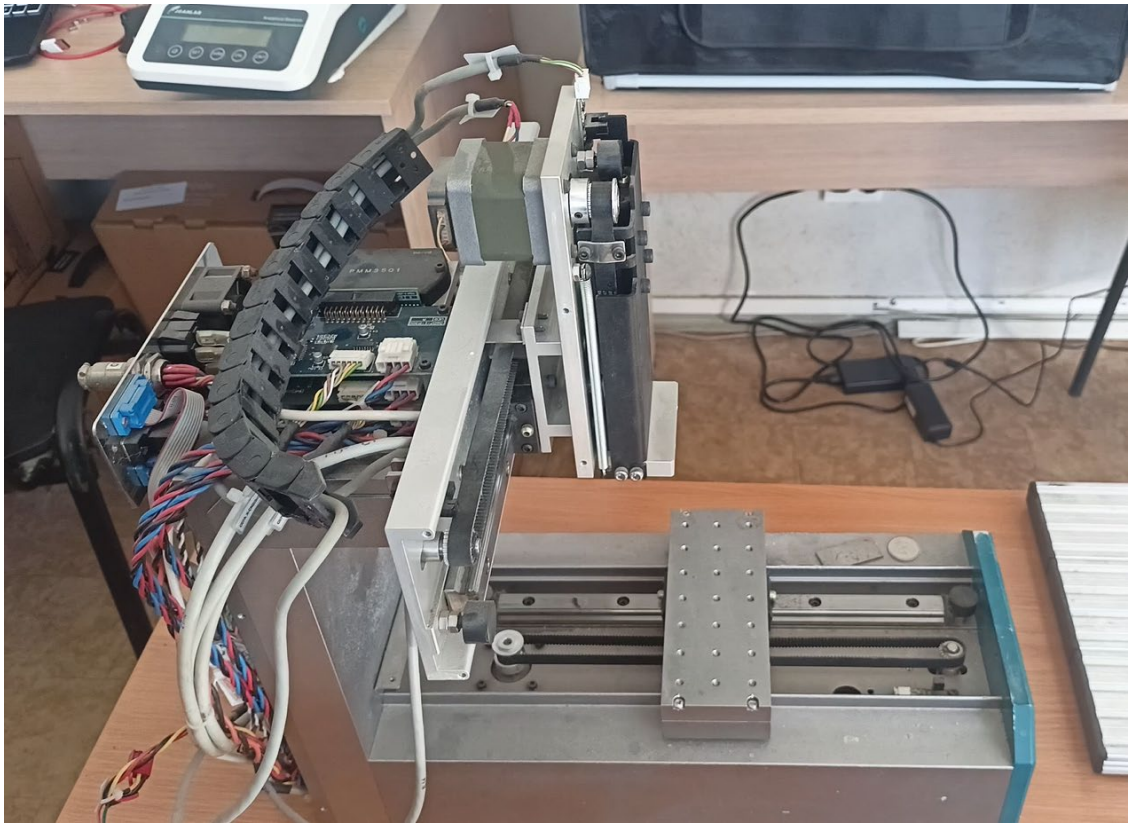

Рисунок 2 – Модифицируемый станок

```python
from math import pi, sin, cos
from typing import Union

class strc:
    def __init__(self, in_="", add_sep=True):
        self.text = in_
        if add_sep and in_:
            self.text += '\r\n'

    def __call__(self, y):
        text = y.text + self.text
        return strc(text, add_sep=False)

header = strc('G21')
home = strc('G28')
home2 = strc('$H')
g90 = strc('$H')
zeroXY = strc('G92X0Y0')
zeroZ = strc('G92Z0')
zeroXYZ = strc('G92X0Y0Z0')

def rect(x, y, w, h):
    return strc('X%2.2f Y%2.2f\r\nX%2.2f Y%2.2f\r\nX%2.2f Y%2.2f\r\nX%2.2f Y%2.2f\r\nX%2.2f Y%2.2f' % (
    x, y, x + w, y, x + w, y + h, x, y + h, x, y))

def set_zero(num):
    return strc(f'G{num}\r\nG90')

def use_ss(num: int):
    if 54 <= num <= 58:
        return strc(f'G{num}')

def gotoxy(**kwargs):
    res = ''
    if not (kwargs.get('fast', None) is None):
        res += f'G0 '
    if not (kwargs.get('slow', None) is None):
        res += f'G1 '
    if not (kwargs.get('vel', None) is None):
        res += f'F{kwargs["vel"]} '
    if not (kwargs.get('x', None) is None):
        res += f'X{kwargs["x"]} '
    if not (kwargs.get('y', None) is None):
        res += f'Y{kwargs["y"]} '
    if not (kwargs.get('z', None) is None):
        res += f'Z{kwargs["z"]} '
    return strc(res)

def pause(sec: float) -> strc:
    return strc(f'G4 P{sec}')

def circle(x: Union[float | int], y: Union[float | int], r: Union[float | int], steps: int, vel: int = 100,
           setf=strc, rounds:float=1.5) -> strc:
    '''Function draw circle in x, y with radius r
    @param x: float - center x coord
    @param y: float - center y coord
    @param r:float - circle radius
    @param steps:int - circle is polygon with steps lines
    @param setf: function|class - strc class with call method
    @return strc object'''

    stp = rounds * 2 * pi / steps
    res = strc()
    for i in range(int(steps)):
        res = setf(
            f"G1 F{vel} " + "X" + str(round(x + r * cos(stp * i), 3)) + " Y" + str(round(y + r * sin(stp * i), 3)))(re
    return res

def str_glue(speed, pause=0):
    return strc(f'M3 S{speed}\r\nG4 P{pause}')

stop_glue = strc('M5\r\n')
```

Рисунок 3 – Код библиотеки функций

```python
from utils import *
from datetime import datetime
# config
length = 4.2 # длина стороны
dx, dy = 11.5, 11.5 # dx - расстояние по x меджу центрами, dy - расстояние по y меджу центрами
stepx, stepy = 10, 10 # stepx - число итераций по оси x (колонок), stepy -  число итераций по оси y (строк)
safeZ = 5
velos = 1500
startX = 10
# program
u = header
#u = home2(u)
u = set_zero(55)(u)
u = zeroXYZ (u)
#u = gotoxy(x=10, y=145, fast=True) (u)
#u = gotoxy(z=48.994, fast=True) (u)
#u = zeroXYZ (u)
u = gotoxy(z=-safeZ, fast=True) (u)
for i in range(stepx):
    for j in range(stepy):
        u = use_ss(55)(u)
        u = gotoxy(x=i*dx, y=j*dy, fast=True) (u)
        u = use_ss(55)(u)
        u = gotoxy(z=0, slow=True, vel=velos)(u)
        u = str_glue(1000, 0) (u)
        u = rect(x=i*dx, y=j*dy, w=length, h=length) (u)
        u = str_glue(0, 0) (u)
        u = gotoxy(z=-safeZ, slow=True, vel=velos)(u)


u = use_ss(54)(u)
u = gotoxy(z=0, fast=True)(u)
#u = gotoxy(x=0, y=0, fast=True)(u)
with open(f'chpu5.gcode', 'w') as f:
    f.write(u.text)
```

Рисунок 4 – Код генератора G-code