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: Тимофеев А. В.

Преподаватель: Ананьевский М.С.

Группа: 3331506/20101

Санкт-Петербург

2025

Содержание

Оглавление

Введение

При изготовлении определённого вида инфракрасных (ИК) датчиков, при их сборке, необходимо наносить силиконовое покрытие на корпуса, затем устанавливать линзу. Корпуса датчиков устанавливаются в заранее подготовленную матрицу, затем матрица монтируется на рабочее поле трёх осевого ЧПУ с дозатором силикона.

Привода осей XYZ – ремённая передача с шаговыми двигателями NEMA17 с углом поворота 1,8 градуса. Дозатором является шприц с пневматической подачей воздуха.

Общая идея управления состоит в исполнении заранее подготовленного алгоритма движения. Во избежание полного цикла разработки программного обеспечение было решено использовать готовые решения с незначительной модификацией. Выбор был основан на программном обеспечении (ПО) GRBL, исполняющем команды движения формата G-code. Программа исполнения формировалась с помощью готового конфигуратора. По концам осей установлены концевые выключатели для относительного позиционирования, так как шаговые двигатели не имеют обратной связи по положению.

В качестве платы управления используется контроллер собственной разработки с чипом atmega328p в основе. Присутствует возможность одновременного управления до 3 шаговых двигателей, 3 вывода чтения концевых выключателей, 3 реле коммутации силовой нагрузки и USB2.0 порт общения с компьютером.

GRBL является лишь исполнителем команды, высылаемых по USB, поэтому необходим визуальный интерфейс на контроллере верхнего уровня, которым может выступить любой ПК на основе Windows или Linux. Выбор остановился на интерфейсе Candle.

1. Модификация прошивки GRBL

Прошивка состоит из нескольких библиотек и файлов, каждый из который отвечает за свою часть работы. К примеру:

- Stepper – воспроизведение импульса управления шаговым двигателем;
- Serial – библиотека общения по USB;
- Report – детектирование и возвращение ошибки при неполадках в работе устройства;
- Grbl – основное тело прошивки, отвечающее за распознавание кода, запрос данных от компьютера по завершении работы команды и тд.

Стандартная конфигурация портов ввода вывода может быть изменена путём редактирования файла cpu_map.h.

Настройки прошивки находятся в файле config.h, но некоторые из них могут быть изменены через командную строку позднее. Для корректной работы устройства необходимо обеспечить раздельную работу 3 модулей реле. Плата управления была спроектирована таким образом, чтобы cpu_map.h не потребовал никаких изменений.

В прошивке исходно предусмотрено управление шпинделем или лазером через сигнал широтно-импульсной модуляции (ШИМ) через вывод D11, сопряжённый с D13. На плате управления, стоят 3 силовых реле, подключенных к портам D11, D13, PC3 контроллера atmega328p. Порт PC3, имеет исходно отдельное управление так как в стандартной конфигурации имеет назначение контроля помпы подачи смазочно-охлаждающей жидкости (СОЖ).

В файле config.h прошивки GRBL необходимо раскомментировать строку #define USE_SPINDLE_DIR_AS_ENABLE_PIN. При этом вывод D13 станет работать отдельно от вывода 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));
```

Закомментрованы 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 предварительно не прошита и у неё отсутствует загрузчик через uart. Согласно документации, для прошивки контроллера сконфигурирован интерфейс SPI и используется USB ASP программатор. В среде ArduinoIDE это проводится следующим образом. Изменённый архив добавляется как библиотека:

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

Редактирование параметров осуществляется через командную строку, через команду “$$” запрашиваются текущие параметры конфигурации. Далее описаны только те параметры, что отличаются от стандартных и команды, меняющие настройку.

$0 = 5 – параметр, отвечающий за минимальную ширину импульса в микросекундах, необходимый для срабатывания электронной схемы. В станке используются драйвера TB6600, у которых, по документации, минимальная длина импульса составляет 3.5 микросекунды.

$1 = 255 – Задержка отключения двигателей в миллисекундах, после прошествия данного времени пропадает напряжение с двигателей. Значение 255 держит шаговые двигатели в постоянно включенном состоянии.

$20 = 1 – программные границы. Настройка безопасности, призванная помочь избежать перемещения за пределы допустимой области. Каждый раз, когда Grbl отправляется G-код движения, он проверяет не произойдет ли выход за пределы допустимой области. И в случае, если происходит нарушение границ, Grbl, где бы он ни находился, немедленно выполняет команду остановки подачи, останавливает шпиндель и охлаждение, а затем выдает сигнал аварии для индикации проблемы. Текущее положение при этом не сбрасывается, поскольку остановка происходит не в результате аварийного принудительного останова, как в случае с жесткими границами.

$22 = 1 – цикл нахождения начальной позиции. Необходимо для детектирования изначально известной точки в пространстве и дальнейшего определения положения других элементов относительно неё.

$100-$102 = 66.667 какое количество шагов двигателя сдвинут каретку на 1мм, рассчитывается по формуле:

$$T_{лп} = \frac{S_{шд} \cdot F_{шд}}{P_{р} \cdot N_{шк}} = \frac{200 \cdot 8}{2 \cdot 12} = 66\frac{2}{3}, \text{где}$$

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 ходе выполнения курсовой работы была проведена модификация и настройка программного обеспечения для управления трёх осевым станком с ЧПУ на базе прошивки GRBL. Основные задачи, поставленные в работе, включали адаптацию прошивки под специфические требования оборудования, настройку параметров движения и формирование G-кода для выполнения конкретных операций.

В результате модификации прошивки GRBL удалось обеспечить корректную работу трёх силовых реле, что позволило управлять шпинделем и другими устройствами независимо друг от друга. Были изменены и оптимизированы параметры управления, что позволило улучшить точность и надёжность работы станка.

Настройка параметров движения и формирование G-кода позволили успешно реализовать алгоритм нанесения силиконового покрытия на корпуса инфракрасных датчиков. Использование интерфейса Candle в качестве управляющей программы обеспечило удобство и простоту взаимодействия с оборудованием.

Таким образом, выполненная работа демонстрирует возможность эффективного использования и адаптации существующих решений для выполнения специфических производственных задач. Полученные результаты могут быть использованы для дальнейшего совершенствования процесса автоматизации производства инфракрасных датчиков.

Список использованных источников

1. AT URL: https://at-machining.com/ru/g-code-cnc/ – свободный доступ (дата обращения: 05.06.2025).
2. CNC-tex URL: https://cnc-tex.ru/news/35/raschet-i-nastroika-remennoi-i-vintovoi-pridachi-chpu.html – свободный доступ (дата обращения: 05.06.2025).
3. Github URL: https://github.com/niXman/mingw-builds-binaries/releases – свободный доступ (дата обращения: 03.06.2025).

Приложения

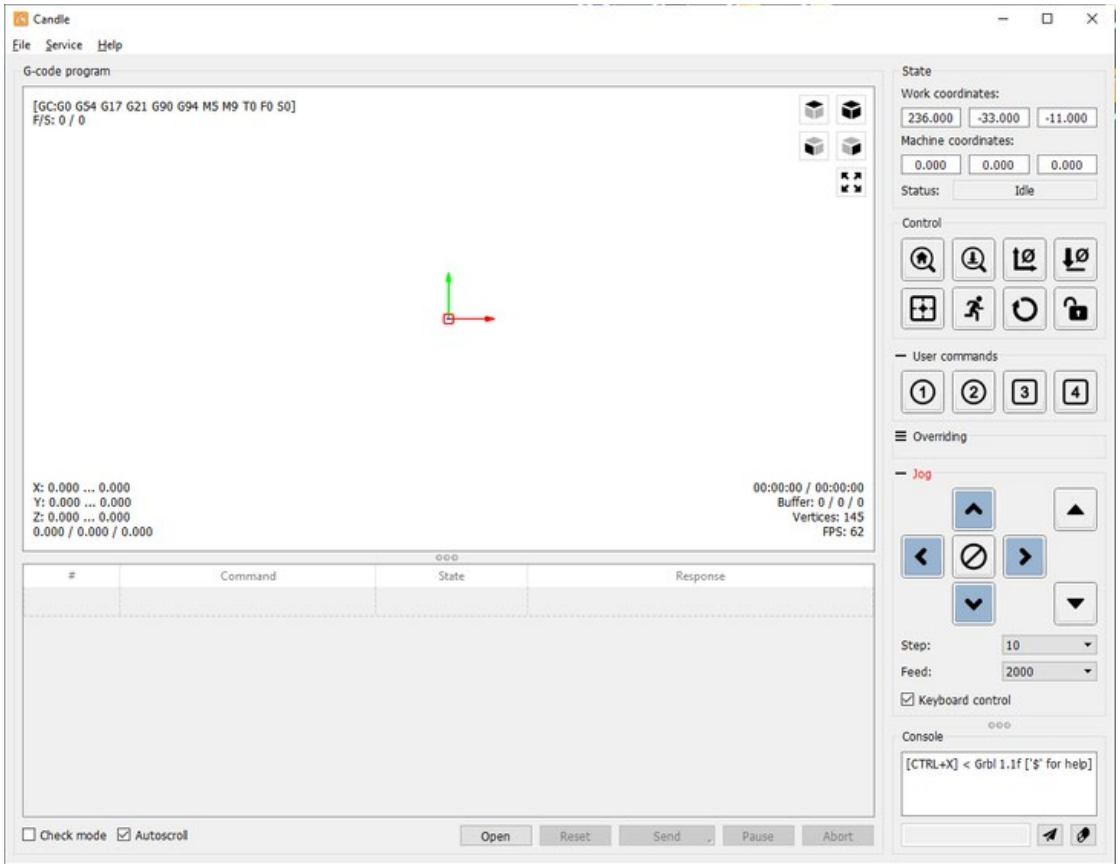

Рисунок 1 – Интерфейс программы candle

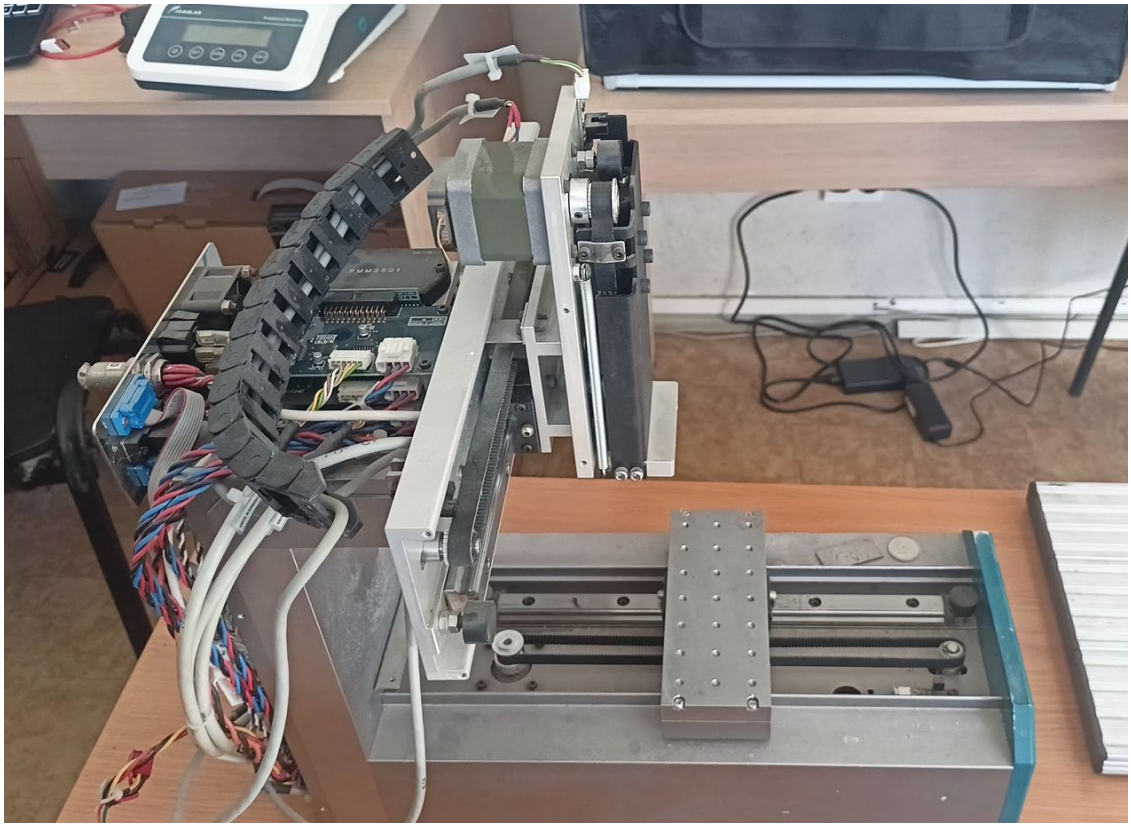

Рисунок 2 – Модифицируемый станок

```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