**ОТЧЕТ**

**«Разработка программно-аппаратного комплекса для модернизации наземной части комплекса управления БпЛА»**

# Структура, функционал и назначение программно-аппаратного комплекса

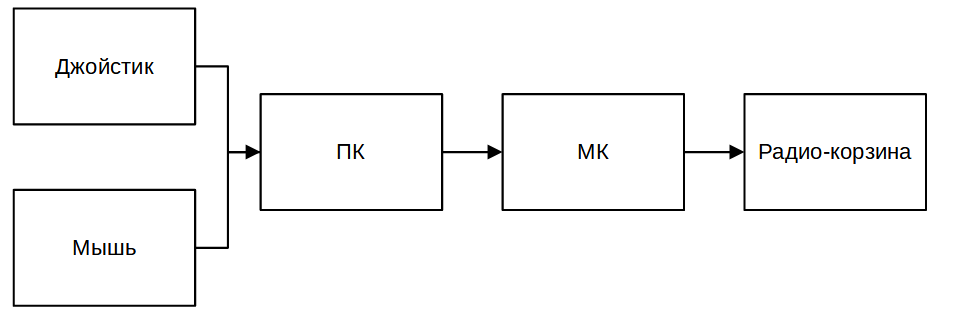
Программно-аппаратный комплекс модернизации системы управления имеет следующий состав:

*Аппаратная часть:*

* персональный компьютер (ПК);
* устройство ввода для взаимодействия с графическим интерфейсом программы (манипулятор типа «мышь»);
* устройство ввода для управления полетом («джойстик»);
* макетная плата на базе микроконтроллера Atmel Mega 324 («Arduino Uno»).

Программная часть:

* системное программное окружение (ОС Windows/Linux, Python3.9, библиотека customtkinter, библиотека serial);
* графическая оболочка «Joy\_fly\_control»;
* программа микроконтроллера (прошивка).



ПК предназначен для подключения джойстика, мыши и микроконтроллера, а также реализации программы «Joy\_fly\_control». Микроконтроллер (МК) предназначен для преобразования и формирования на своих выводах управляющих воздействий с ПК в сигналы PWM или S-BUS.

Программа «Joy\_fly\_control» предназначена для осуществления управления полетом в ручном режиме, предоставляя в этих целях пользовательский интерфейс для конфигурирования настроек и управления процессом чтения-передачи данных. Для этого она выполняет следующие функции:

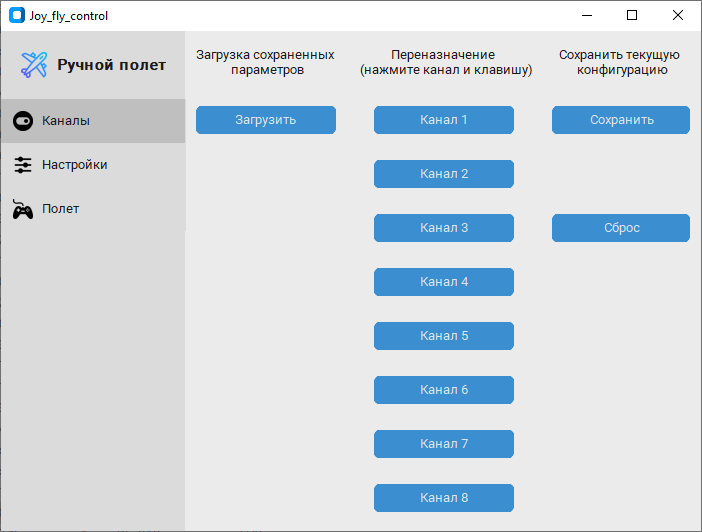
* обнаружение подключенных устройств ввода команд (джойстиков);
* осуществление привязки клавиш джойстика к каналам управления;
* осуществление настройки кривой отклика (чувствительности в заданном диапазоне) стиков для каждого канала;
* сохранение и загрузку настроек привязки клавиш к каналам и кривых отклика по всем каналам;
* управление потоками чтения/записи аппаратных частей программно-аппаратного комплекса;
* чтение событий нажатия клавиш джойстика, назначенных по каналам, считывание положений стиков и осуществление масштабирования в соответствии с кривой чувствительности соответствующего канала;
* передача масштабированных в соответствии с кривой отклика значений стиков в последовательный порт.

# Описание графического интерфейса программы

При запуске программы «Joy\_fly\_control» появляется главное окно, состоящее из окна навигации в левой части главного окна и трех вкладок:

* «Каналы»;
* «Настройки»;
* «Полет».

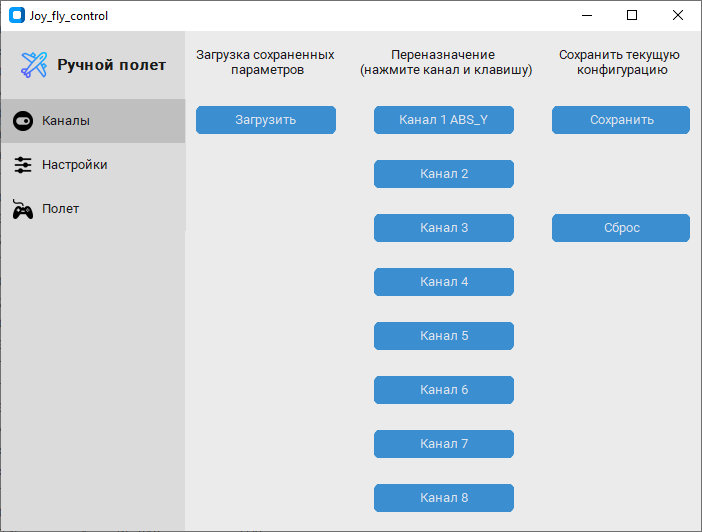
Окно программы при запуске показано на рисунке ниже.



Окно программы «Joy\_fly\_control» после запуска, открыта вкладка конфигурирования каналов

Окно «Каналы» предоставляет пользователю возможность осуществить привязку каналов к осям стиков и клавишам подключенного джойстика.

Процесс привязки каналов к клавишам осуществляется следующим образом. Чтобы привязать канал к клавише необходимо нажать кнопку соответствующего канала левой кнопкой мыши. После этого кнопка с номером канала изменит цвет, и программа будет ожидать события нажатия клавиши джойстика. Пользователю необходимо нажать нужную клавишу джойстика, после чего кнопка с номером канала изменит цвет на первоначальный, а в ее названии помимо номера канала появится название привязанной клавиши или оси стика джойстика.



Каналу 1 назначен ввод с оси Y левого стика джойстика

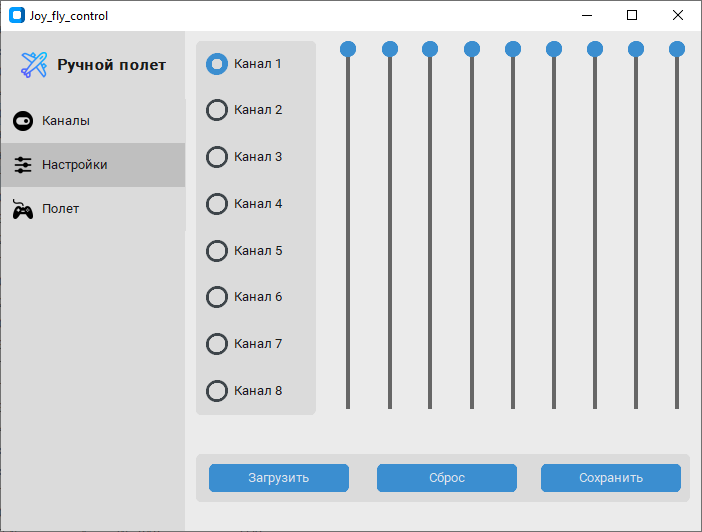
При необходимости можно переназначить клавишу для любого канала повторением вышеописанных действий.

Привязка всех каналов может быть сброшена нажатием кнопки «Сброс».

После настройки всех каналов, текущие привязки могут быть сохранены нажатием кнопки «Сохранить». Настройки храниться на жестком диске в папке с программой в файле «channel\_settings.npy».

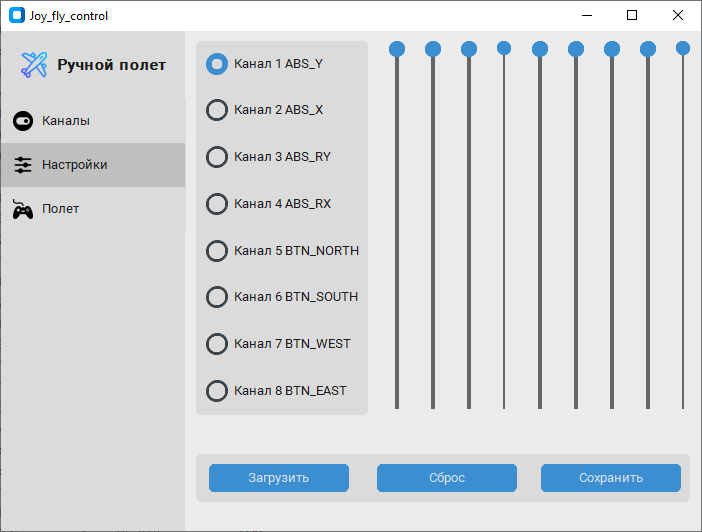
Сохраненные настройки конфигурации каналов могут быть загружены нажатием кнопки «Загрузить».

После конфигурирования каналов можно перейти к настройке чувствительности стиков по осям, перейдя в навигационном окне слева на вкладку «Настройки». Если каналы не сконфигурированы, можно произвести настройки с привязкой по номеру канала, которые после привязки в окне «Каналы» будут применены к данным с соответствующего выхода джойстика.



Вкладка конфигурирования кривой отклика стиков, - каналы не привязаны к клавишам джойстика

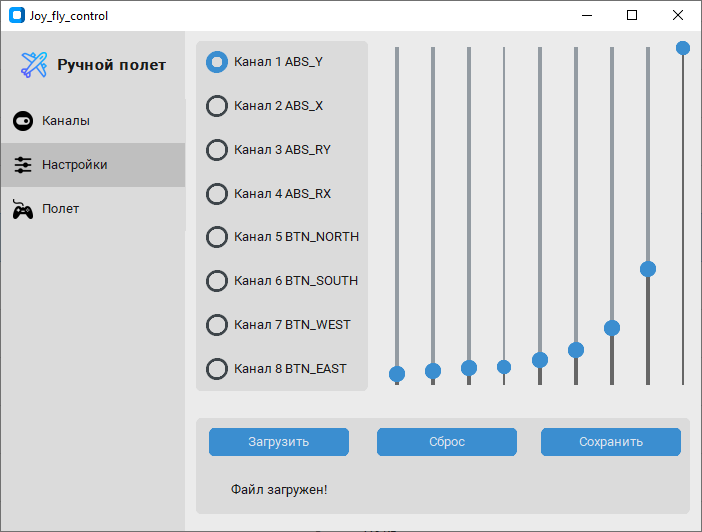
Если каналы сконфигурированы, то информация о назначенных каналах будет отображаться, как показано на рисунке ниже.



Вкладка конфигурирования кривой отклика стиков, - каналы привязаны к клавишам джойстика

Вкладка «Настройки» имеет три зоны: зона выбора конфигурируемого канала, зона с 9 подвижными движками для настройки, зона с кнопками «Загрузить», «Сброс», «Сохранить».

Пример настройки кривой отклика канала 1 показан на рисунке ниже.



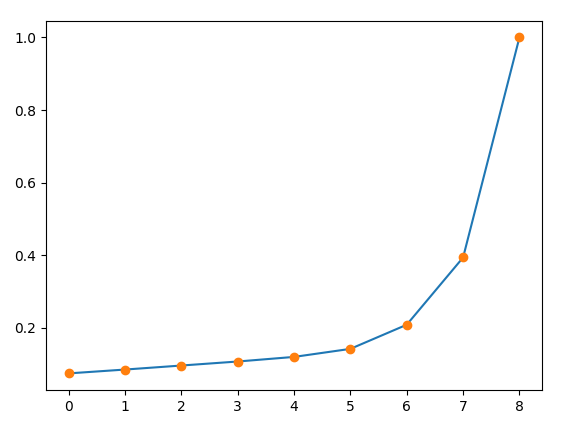
Пример настройки канала 1

Настройки чувствительности каждого канала хранятся в двумерной матрице (таблице) self.stick\_sensity. Размер матрицы составляет 8 строк (по числу каналов) на 9 столбцов (по числу движков в окне настройки). Для примера на рисунке выше значения движков в составляют:

stick\_sensity[0, :] = [0.07486631, 0.08544304, 0.09651899, 0.10759494, 0.12025316, 0.14240506, 0.20886076, 0.39398734, 1]

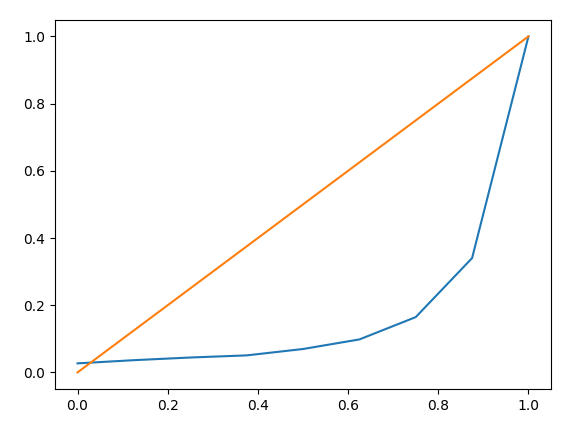
Настройка чувствительности организована следующим образом: весь диапазон стика разбивается на две части от среднего положения. Каждый поддиапазон разбивается на 8 равных участков, за настройку которых отвечают движки в окне настройки. После настройки, значения движков (9 шт.) каждого из каналов подвергаются полиномиальной аппроксимации.

На рисунке ниже оранжевыми точками показаны значения движков, а синей линией показана аппроксимирующая кривая.



В дальнейшем, при работе выходной сигнал джойстика каждого поддиапазона подвергается преобразованию в соответствии с функцией, аппроксимирующей настройки чувствительности для данного канала.

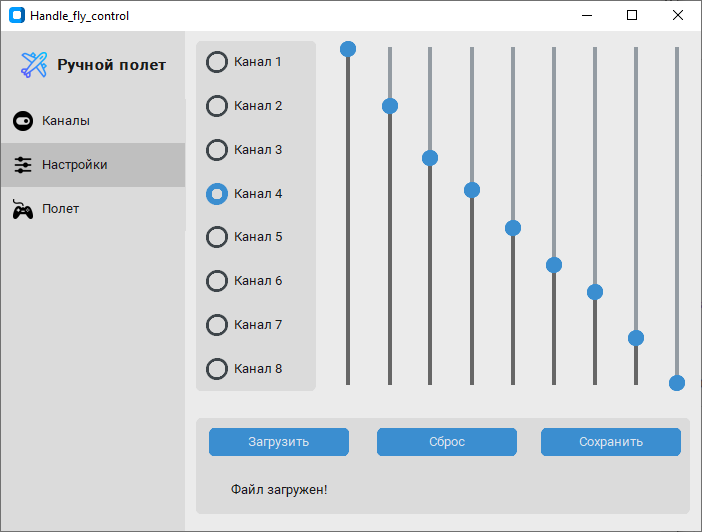
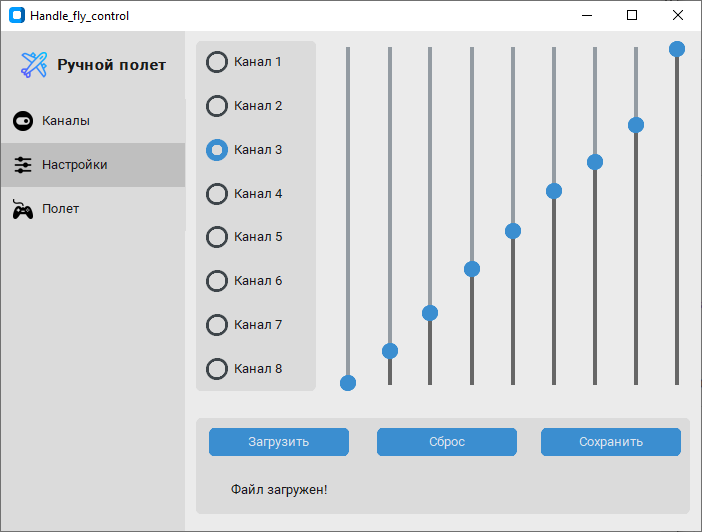
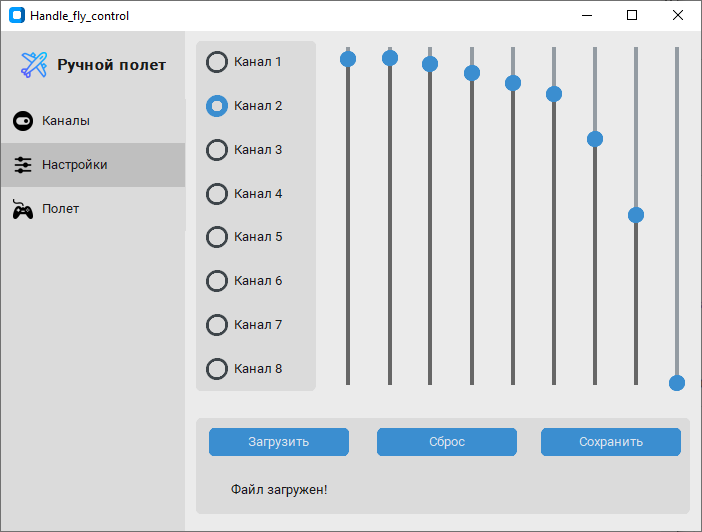
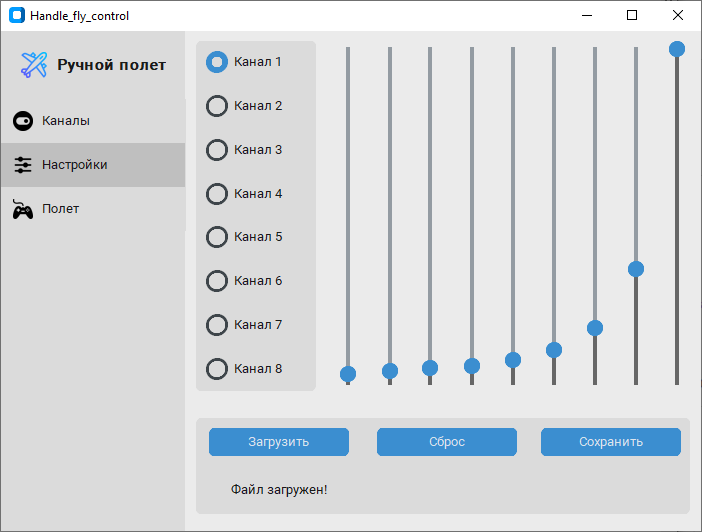
Сравнение исходной и скорректированной кривой отклика показано на рисунке ниже.

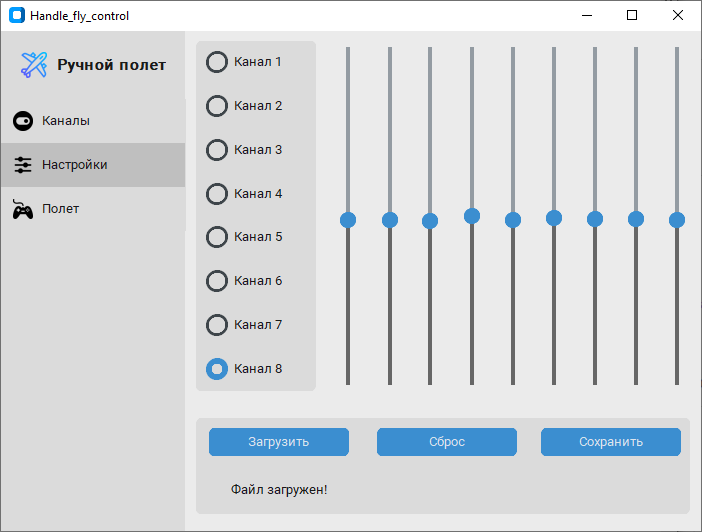
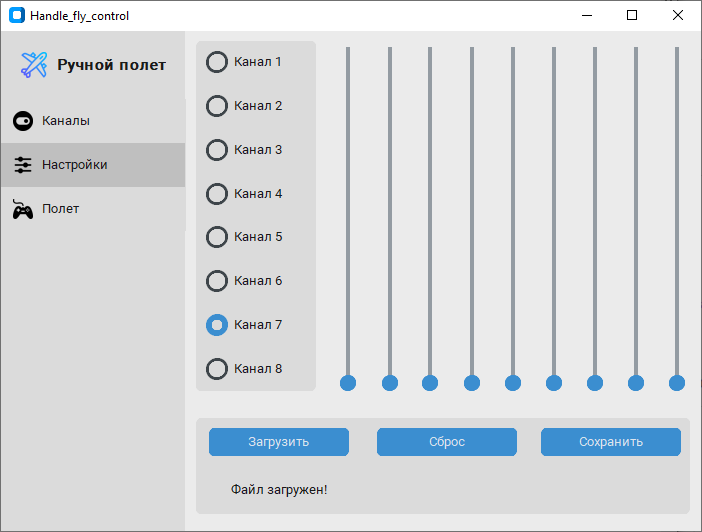
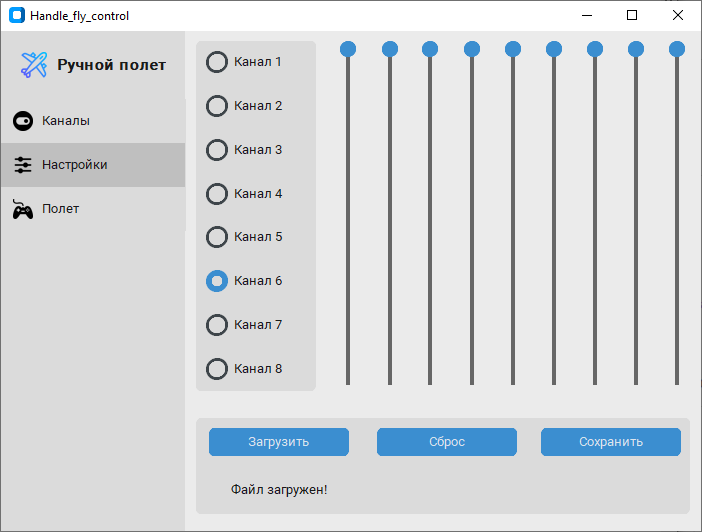
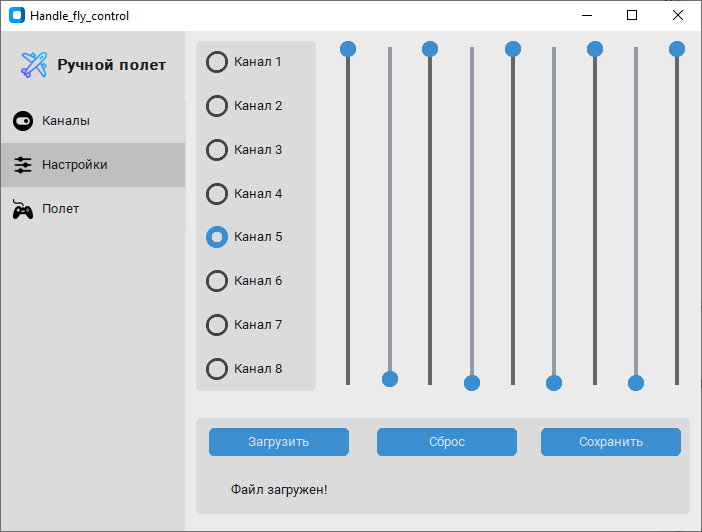


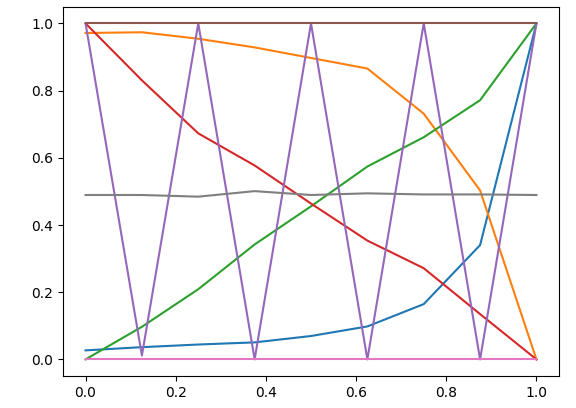
Сравнение исходной (линейной, -оранжевая линия) и скорректированной (голубая линия) кривой отклика стика

Значения stick\_sensity изменяются путем выбора канала и настройкой уровней. Настройка уровней и сброс параметров производятся только для выбранного канала, а сохранение настроек производится для всех каналов единовременно.

На рисунке ниже показаны примеры настройки кривой отклика.



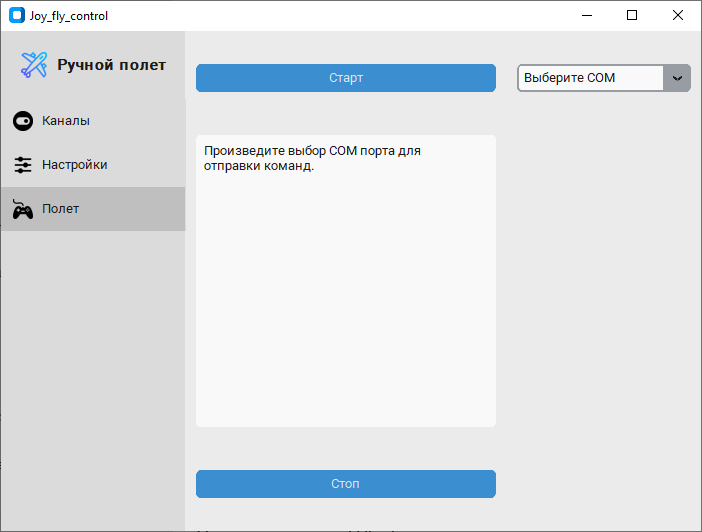




Примеры настроек кривой отклика и соответствующие им функции аппроксимации

После конфигурирования настройки могут быть сохранены кнопкой «Сохранение» и в дальнейшем загружены кнопкой «Загрузить».

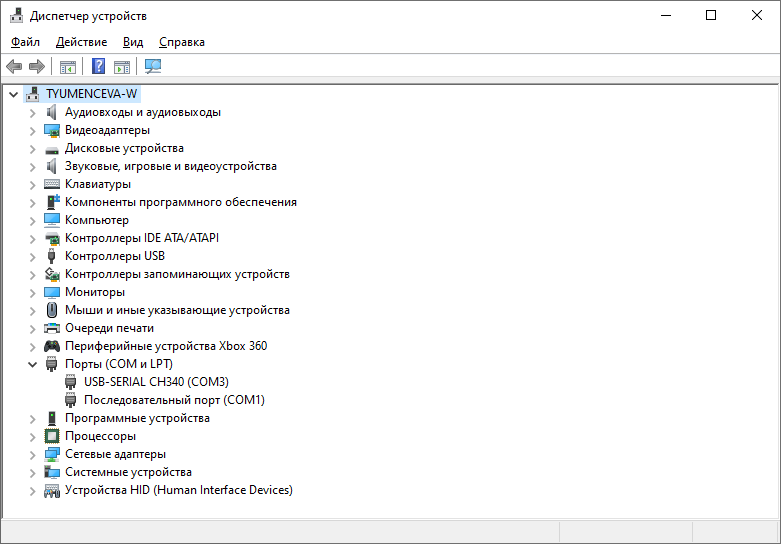
После настройки привязки каналов и чувствительности можно преходить к запуску потока считывания, масштабирования и предачи данных на микроконтроллер (в последовательный порт) на вкладке «Полет».



Окно «Полет» после перехода на него в области навигации

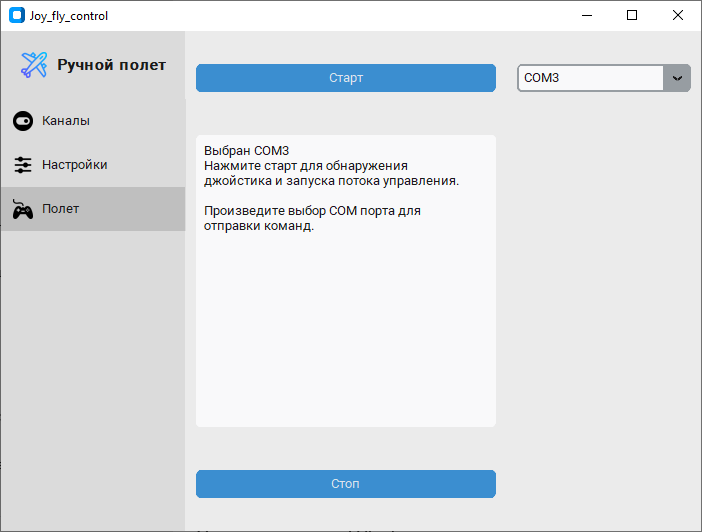
Окно «Полет» имеет клавиши «Старт» / «Стоп», область вывода пользовательских сообщений, а также выпадающее меню для выбора СОМ порта, в который будут отправляться команды.

Для начала ручного управления необходимо в выпадающем меню справа выбрать СОМ порт, к которому подключена плата микроконтроллера. Программа сканирует список доступных к открытию СОМ портов при запуске, а также при переходе на вкладку «Полет». Таким образом, в выпадающем списке будут содержаться все доступные к подключению СОМ порты. Для выбора порта, к которому подключено устройство можно воспользоваться диспетчером устройств.

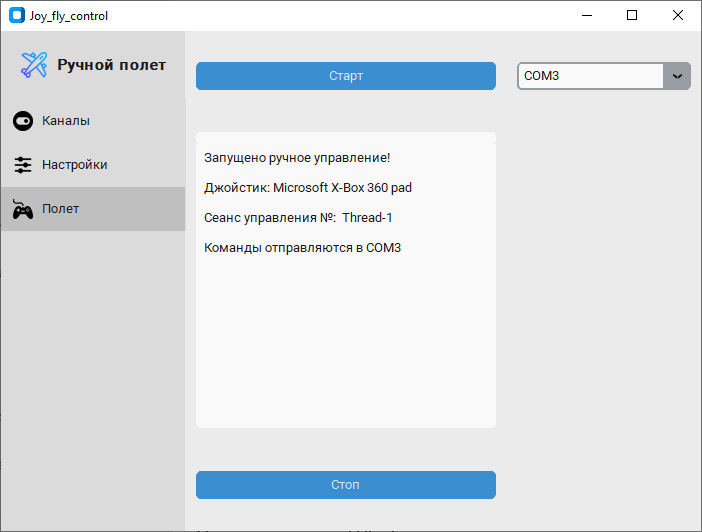


Микроконтроллер подключен к СОМ3

После выбора порта программа в диалоговом окне сообщит о том, что сделан выбор порта и предложит нажать кнопку старт.

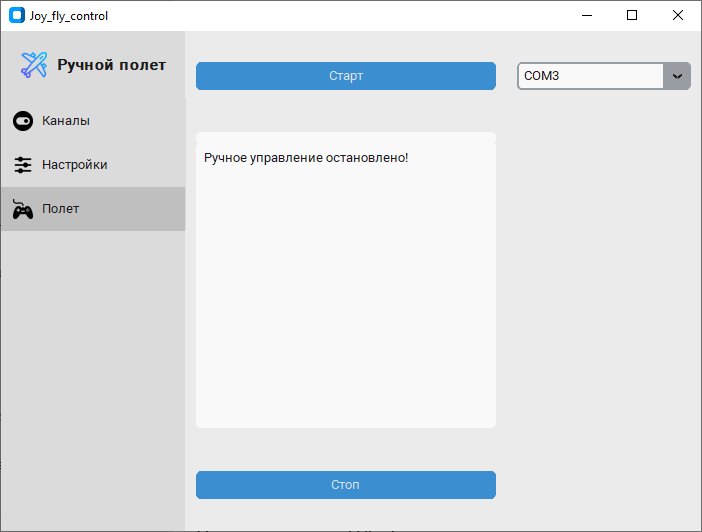


После нажатия кнопки «Старт» программа запустит отдельный вычислительный поток, в котором производится считывание сигналов (событий нажатия клавиш или изменения положения стиков по осям) с джойстика, их фильтрация в соответствии с привязкой к каналам, произведенной в окне «Каналы», масштабирование в соответствии с настройками для каналов, произведенными в окне «Настройки» и передача масштабированных значений с номером канала в последовательный порт.



Вид окна после нажатия кнопки «Старт»

При нажатии кнопки «Стоп», поток считывания преобразования и отправки данных будет остановлен.



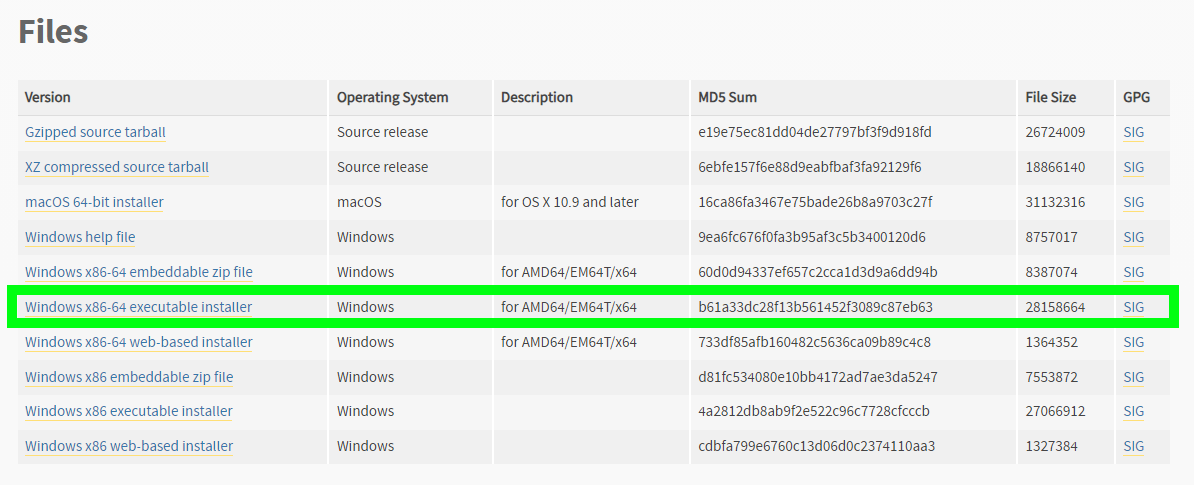
# Установка

Для Windows

Установка пакетов для работы и первичное конфигурирование (требует подключения к сети Internet).

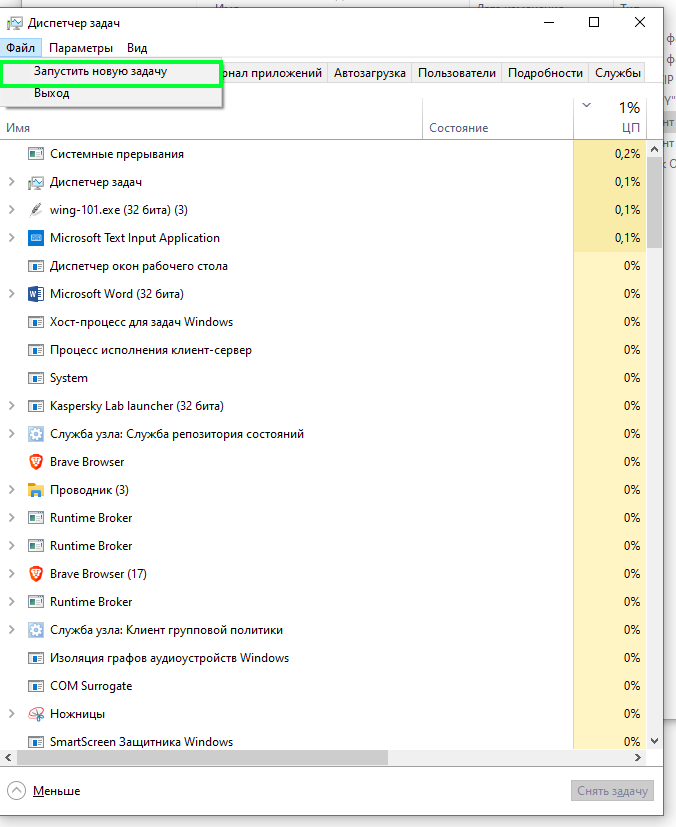
1. Скачать и установить Python3.9

<https://www.python.org/downloads/release/python-390/>

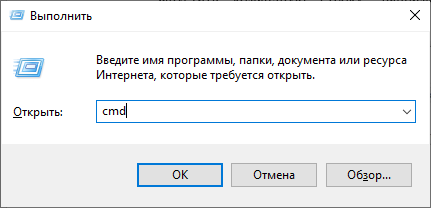


1. Установить библиотеку inputs

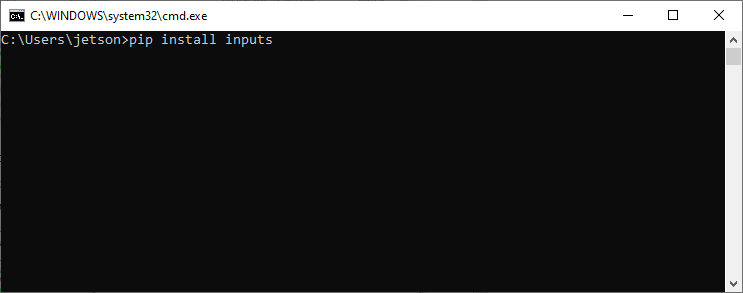
Запустить командную строку, например, сочетанием клавиш «Windows» + «R», или из диспетчера задач.



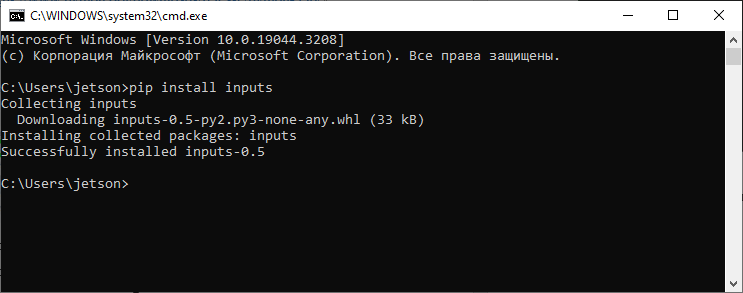
Ввести в командной строке команду: *Cmd*



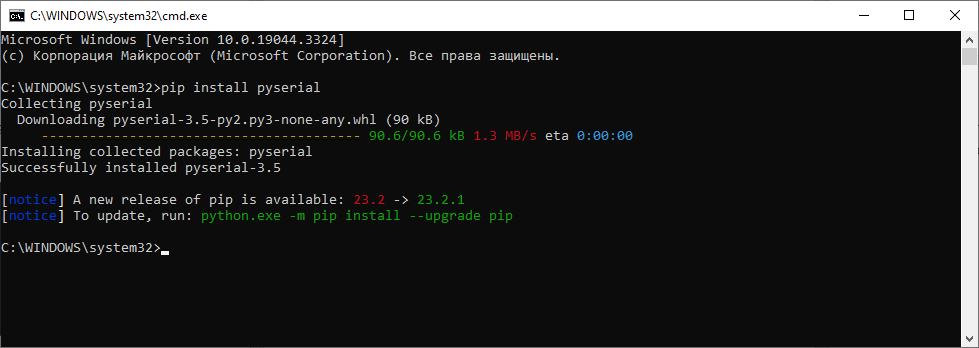
В окне cmd.exe ввести команду: *pip install inputs*



Нажать Enter и дождаться сообщения о завершении установки: *Successfully installed inputs*



pip install pyserial



Под windows джойстик «logitech gamepad f310» нужно перевести в режим «X» переключателем на обратной стороне джойстика.

# Код программы «Joy\_fly\_control»

from \_\_future\_\_ import print\_function

import customtkinter

import os

from PIL import Image

import inputs\_lib

import threading

from threading import Thread

import time

import serial

import tkinter

import numpy as np

import ctypes

import platform

import serial

class App(customtkinter.CTk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.WIN = True if platform.system() == 'Windows' else False

self.NIX = True if platform.system() == 'Linux' else False

self.trig = 1

self.channel\_list = ['', '', '', '', '', '', '', '']

self.stick\_sensity = np.array([[1., 1., 1., 1., 1., 1., 1., 1., 1.],

[1., 1., 1., 1., 1., 1., 1., 1., 1.],

[1., 1., 1., 1., 1., 1., 1., 1., 1.],

[1., 1., 1., 1., 1., 1., 1., 1., 1.],

[1., 1., 1., 1., 1., 1., 1., 1., 1.],

[1., 1., 1., 1., 1., 1., 1., 1., 1.],

[1., 1., 1., 1., 1., 1., 1., 1., 1.],

[1., 1., 1., 1., 1., 1., 1., 1., 1.]])

self.aprox\_models = np.ones((8, 10), dtype='float')

self.com\_port = ''

self.title("Joy\_fly\_control")

self.geometry("700x500")

# set grid layout 1x2

self.grid\_rowconfigure(0, weight=1)

self.grid\_columnconfigure(1, weight=1)

# load images with light and dark mode image

image\_path = os.path.join(os.path.dirname(

os.path.realpath(\_\_file\_\_)), "test\_images")

self.logo\_image = customtkinter.CTkImage(Image.open(os.path.join(

image\_path, "CustomTkinter\_logo\_single.png")), size=(26, 26))

self.image\_icon\_image = customtkinter.CTkImage(Image.open(

os.path.join(image\_path, "image\_icon\_light.png")), size=(20, 20))

self.joystik\_image = customtkinter.CTkImage(light\_image=Image.open(os.path.join(image\_path, "joystik\_dark.png")),

dark\_image=Image.open(os.path.join(image\_path, "joystik\_light.png")), size=(20, 20))

self.chat\_image = customtkinter.CTkImage(light\_image=Image.open(os.path.join(image\_path, "chat\_dark.png")),

dark\_image=Image.open(os.path.join(image\_path, "chat\_light.png")), size=(20, 20))

self.add\_user\_image = customtkinter.CTkImage(light\_image=Image.open(os.path.join(image\_path, "add\_user\_dark.png")),

dark\_image=Image.open(os.path.join(image\_path, "add\_user\_light.png")), size=(20, 20))

# create navigation frame

self.navigation\_frame = customtkinter.CTkFrame(self, corner\_radius=0)

self.navigation\_frame.grid(row=0, column=0, sticky="nsew")

self.navigation\_frame.grid\_rowconfigure(4, weight=1)

self.navigation\_frame\_label = customtkinter.CTkLabel(self.navigation\_frame, text=" Ручной полет", image=self.logo\_image,

compound="left", font=customtkinter.CTkFont(size=15, weight="bold"))

self.navigation\_frame\_label.grid(row=0, column=0, padx=20, pady=20)

self.joystik\_button = customtkinter.CTkButton(self.navigation\_frame, corner\_radius=0, height=40, border\_spacing=10, text="Полет",

fg\_color="transparent", text\_color=("gray10", "gray90"), hover\_color=("gray70", "gray30"),

image=self.joystik\_image, anchor="w", command=self.joystik\_button\_event)

self.joystik\_button.grid(row=3, column=0, sticky="ew")

self.chanel\_selector\_button = customtkinter.CTkButton(self.navigation\_frame, corner\_radius=0, height=40, border\_spacing=10, text="Каналы",

fg\_color="transparent", text\_color=("gray10", "gray90"), hover\_color=("gray70", "gray30"),

image=self.chat\_image, anchor="w", command=self.chanel\_selector\_button\_event)

self.chanel\_selector\_button.grid(row=1, column=0, sticky="ew")

self.settings\_button = customtkinter.CTkButton(self.navigation\_frame, corner\_radius=0, height=40, border\_spacing=10, text="Настройки",

fg\_color="transparent", text\_color=("gray10", "gray90"), hover\_color=("gray70", "gray30"),

image=self.add\_user\_image, anchor="w", command=self.settings\_button\_event)

self.settings\_button.grid(row=2, column=0, sticky="ew")

# create joystik frame

self.joystik\_frame = customtkinter.CTkFrame(

self, corner\_radius=0, fg\_color="transparent")

self.joystik\_frame.grid\_columnconfigure(0, weight=3)

self.joystik\_frame.grid\_columnconfigure(1, weight=1)

self.joystik\_frame.grid\_rowconfigure(0, weight=1)

self.joystik\_frame.grid\_rowconfigure(1, weight=2)

self.joystik\_frame.grid\_rowconfigure(2, weight=1)

self.joystik\_frame\_button\_1 = customtkinter.CTkButton(

self.joystik\_frame, text="Старт", command=self.button\_start\_event)

self.joystik\_frame\_button\_1.grid(

row=0, column=0, padx=10, pady=10, sticky="ew")

self.textbox = customtkinter.CTkTextbox(self.joystik\_frame)

self.textbox.grid(row=1, column=0, padx=(

10, 10), pady=(10, 10), sticky="nsew")

if self.com\_port == '':

self.textbox.insert(

"0.0", "Произведите выбор СОМ порта для \nотправки команд.\n\n")

else:

self.textbox.insert(

"0.0", 'Выбран ' + self.com\_port + "\nНажмите старт для обнаружения \nджойстика и запуска потока управления.\n\n")

self.joystik\_frame\_button\_2 = customtkinter.CTkButton(

self.joystik\_frame, text="Стоп", command=self.button\_stop\_event)

self.joystik\_frame\_button\_2.grid(

row=2, column=0, padx=10, pady=10, sticky="ew")

self.combobox\_com = customtkinter.CTkComboBox(

self.joystik\_frame, values=self.scan\_com(), command=self.combobox\_com\_event)

self.combobox\_com.grid(

row=0, column=1, padx=10, pady=10, sticky="ew")

self.combobox\_com.set("Выберите СОМ")

# create second frame

self.second\_frame = customtkinter.CTkFrame(

self, corner\_radius=0, fg\_color="transparent")

self.second\_frame.grid\_columnconfigure(0, weight=1)

self.second\_frame.grid\_columnconfigure(1, weight=1)

self.second\_frame.grid\_columnconfigure(2, weight=1)

self.second\_frame.grid\_rowconfigure(

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9], weight=1)

self.second\_frame\_label\_0 = customtkinter.CTkLabel(

self.second\_frame, text='Загрузка сохраненных \n параметров', width=120, height=25, fg\_color="transparent", corner\_radius=0)

self.second\_frame\_label\_0.grid(row=0, column=0, padx=10, pady=10)

self.second\_frame\_label\_1 = customtkinter.CTkLabel(

self.second\_frame, text='Переназначение \n (нажмите канал и клавишу)', width=120, height=25, fg\_color="transparent", corner\_radius=0)

self.second\_frame\_label\_1.grid(row=0, column=1, padx=10, pady=13)

self.second\_frame\_label\_2 = customtkinter.CTkLabel(

self.second\_frame, text='Сохранить текущую \n конфигурацию', width=120, height=25, fg\_color="transparent", corner\_radius=0)

self.second\_frame\_label\_2.grid(row=0, column=2, padx=10, pady=10)

self.second\_frame\_btn\_download = customtkinter.CTkButton(

self.second\_frame, text="Загрузить", command=self.btn\_download\_event)

self.second\_frame\_btn\_download.grid(

row=1, column=0, padx=10, pady=10)

self.second\_frame\_btn\_ch\_1 = customtkinter.CTkButton(

self.second\_frame, text="Канал 1", command=self.btn\_ch\_1\_event)

self.second\_frame\_btn\_ch\_1.grid(row=1, column=1, padx=10, pady=10)

self.second\_frame\_btn\_ch\_2 = customtkinter.CTkButton(

self.second\_frame, text="Канал 2", command=self.btn\_ch\_2\_event)

self.second\_frame\_btn\_ch\_2.grid(row=2, column=1, padx=10, pady=10)

self.second\_frame\_btn\_ch\_3 = customtkinter.CTkButton(

self.second\_frame, text="Канал 3", command=self.btn\_ch\_3\_event)

self.second\_frame\_btn\_ch\_3.grid(row=3, column=1, padx=10, pady=10)

self.second\_frame\_btn\_ch\_4 = customtkinter.CTkButton(

self.second\_frame, text="Канал 4", command=self.btn\_ch\_4\_event)

self.second\_frame\_btn\_ch\_4.grid(row=4, column=1, padx=10, pady=10)

self.second\_frame\_btn\_ch\_5 = customtkinter.CTkButton(

self.second\_frame, text="Канал 5", command=self.btn\_ch\_5\_event)

self.second\_frame\_btn\_ch\_5.grid(row=5, column=1, padx=10, pady=10)

self.second\_frame\_btn\_ch\_6 = customtkinter.CTkButton(

self.second\_frame, text="Канал 6", command=self.btn\_ch\_6\_event)

self.second\_frame\_btn\_ch\_6.grid(row=6, column=1, padx=10, pady=10)

self.second\_frame\_btn\_ch\_7 = customtkinter.CTkButton(

self.second\_frame, text="Канал 7", command=self.btn\_ch\_7\_event)

self.second\_frame\_btn\_ch\_7.grid(row=7, column=1, padx=10, pady=10)

self.second\_frame\_btn\_ch\_8 = customtkinter.CTkButton(

self.second\_frame, text="Канал 8", command=self.btn\_ch\_8\_event)

self.second\_frame\_btn\_ch\_8.grid(row=8, column=1, padx=10, pady=10)

self.second\_frame\_btn\_save = customtkinter.CTkButton(

self.second\_frame, text="Сохранить", command=self.btn\_save\_event)

self.second\_frame\_btn\_save.grid(row=1, column=2, padx=10, pady=10)

self.second\_frame\_btn\_reset = customtkinter.CTkButton(

self.second\_frame, text="Сброс", command=self.btn\_reset\_event)

self.second\_frame\_btn\_reset.grid(row=3, column=2, padx=10, pady=10)

# create third frame

self.third\_frame = customtkinter.CTkFrame(

self, corner\_radius=0, fg\_color="transparent")

self.third\_frame.grid\_columnconfigure(

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9], weight=1)

self.third\_frame.grid\_rowconfigure(0, weight=1)

self.third\_frame.grid\_rowconfigure(1, weight=1)

self.channel\_frame = customtkinter.CTkFrame(self.third\_frame)

self.channel\_frame.grid(

row=0, column=0, padx=10, pady=(10, 10), sticky="nsw")

self.channel\_frame.grid\_columnconfigure(1, weight=1)

self.channel\_frame.grid\_rowconfigure(

[0, 1, 2, 3, 4, 5, 6, 7], weight=1)

self.radiobutton\_var = customtkinter.IntVar(value=0)

self.radiobutton\_1 = customtkinter.CTkRadioButton(

self.channel\_frame, text='Канал 1' + self.channel\_list[0], command=self.radiobutton\_event, variable=self.radiobutton\_var, value=0)

self.radiobutton\_1.grid(

row=0, column=0, padx=10, pady=10, sticky="w")

self.radiobutton\_2 = customtkinter.CTkRadioButton(

self.channel\_frame, text='Канал 2', command=self.radiobutton\_event, variable=self.radiobutton\_var, value=1)

self.radiobutton\_2.grid(

row=1, column=0, padx=10, pady=10, sticky="w")

self.radiobutton\_3 = customtkinter.CTkRadioButton(

self.channel\_frame, text='Канал 3', command=self.radiobutton\_event, variable=self.radiobutton\_var, value=2)

self.radiobutton\_3.grid(

row=2, column=0, padx=10, pady=10, sticky="w")

self.radiobutton\_4 = customtkinter.CTkRadioButton(

self.channel\_frame, text='Канал 4', command=self.radiobutton\_event, variable=self.radiobutton\_var, value=3)

self.radiobutton\_4.grid(

row=3, column=0, padx=10, pady=10, sticky="w")

self.radiobutton\_5 = customtkinter.CTkRadioButton(

self.channel\_frame, text='Канал 5', command=self.radiobutton\_event, variable=self.radiobutton\_var, value=4)

self.radiobutton\_5.grid(

row=4, column=0, padx=10, pady=10, sticky="w")

self.radiobutton\_6 = customtkinter.CTkRadioButton(

self.channel\_frame, text='Канал 6', command=self.radiobutton\_event, variable=self.radiobutton\_var, value=5)

self.radiobutton\_6.grid(

row=5, column=0, padx=10, pady=10, sticky="w")

self.radiobutton\_7 = customtkinter.CTkRadioButton(

self.channel\_frame, text='Канал 7', command=self.radiobutton\_event, variable=self.radiobutton\_var, value=6)

self.radiobutton\_7.grid(

row=6, column=0, padx=10, pady=10, sticky="w")

self.radiobutton\_8 = customtkinter.CTkRadioButton(

self.channel\_frame, text='Канал 8', command=self.radiobutton\_event, variable=self.radiobutton\_var, value=7)

self.radiobutton\_8.grid(

row=7, column=0, padx=10, pady=10, sticky="w")

self.slider\_1 = customtkinter.CTkSlider(

self.third\_frame, command=self.slider\_1\_callback, from\_=0, to=1, orientation="vertical")

self.slider\_1.grid(row=0, column=1, padx=10, pady=10, sticky='nsw')

self.slider\_1.set(self.stick\_sensity[self.radiobutton\_var.get(), 0])

self.slider\_2 = customtkinter.CTkSlider(

self.third\_frame, command=self.slider\_2\_callback, from\_=0, to=1, orientation="vertical")

self.slider\_2.grid(row=0, column=2, padx=10, pady=10, sticky='nsw')

self.slider\_2.set(self.stick\_sensity[self.radiobutton\_var.get(), 1])

self.slider\_3 = customtkinter.CTkSlider(

self.third\_frame, command=self.slider\_3\_callback, from\_=0, to=1, orientation="vertical")

self.slider\_3.grid(row=0, column=3, padx=10, pady=10, sticky='nsw')

self.slider\_3.set(self.stick\_sensity[self.radiobutton\_var.get(), 2])

self.slider\_4 = customtkinter.CTkSlider(

self.third\_frame, command=self.slider\_4\_callback, from\_=0, to=1, orientation="vertical")

self.slider\_4.grid(row=0, column=4, padx=10, pady=10, sticky='nsw')

self.slider\_4.set(self.stick\_sensity[self.radiobutton\_var.get(), 3])

self.slider\_5 = customtkinter.CTkSlider(

self.third\_frame, command=self.slider\_5\_callback, from\_=0, to=1, orientation="vertical")

self.slider\_5.grid(row=0, column=5, padx=10, pady=10, sticky='nsw')

self.slider\_5.set(self.stick\_sensity[self.radiobutton\_var.get(), 4])

self.slider\_6 = customtkinter.CTkSlider(

self.third\_frame, command=self.slider\_6\_callback, from\_=0, to=1, orientation="vertical")

self.slider\_6.grid(row=0, column=6, padx=10, pady=10, sticky='nsw')

self.slider\_6.set(self.stick\_sensity[self.radiobutton\_var.get(), 5])

self.slider\_7 = customtkinter.CTkSlider(

self.third\_frame, command=self.slider\_7\_callback, from\_=0, to=1, orientation="vertical")

self.slider\_7.grid(row=0, column=7, padx=10, pady=10, sticky='nsw')

self.slider\_7.set(self.stick\_sensity[self.radiobutton\_var.get(), 6])

self.slider\_8 = customtkinter.CTkSlider(

self.third\_frame, command=self.slider\_8\_callback, from\_=0, to=1, orientation="vertical")

self.slider\_8.grid(row=0, column=8, padx=10, pady=10, sticky='nsw')

self.slider\_8.set(self.stick\_sensity[self.radiobutton\_var.get(), 7])

self.slider\_9 = customtkinter.CTkSlider(

self.third\_frame, command=self.slider\_9\_callback, from\_=0, to=1, orientation="vertical")

self.slider\_9.grid(row=0, column=9, padx=10, pady=10, sticky='nsw')

self.slider\_9.set(self.stick\_sensity[self.radiobutton\_var.get(), 8])

self.button\_frame = customtkinter.CTkFrame(self.third\_frame)

self.button\_frame.grid(

row=1, column=0, padx=10, pady=(10, 10), sticky="ew", columnspan=10)

self.button\_frame.grid\_columnconfigure(

[self.radiobutton\_var.get(), 1], weight=1)

self.button\_frame.grid\_rowconfigure(

[0, 1], weight=1)

# Дописать функции вызова

self.third\_frame\_btn\_load = customtkinter.CTkButton(

self.button\_frame, text="Загрузить", command=self.btn\_load\_1\_event)

self.third\_frame\_btn\_load.grid(

row=0, column=0, padx=10, pady=(10, 10))

# Дописать функции вызова

self.third\_frame\_btn\_reset = customtkinter.CTkButton(

self.button\_frame, text="Сброс", command=self.btn\_reset\_1\_event)

self.third\_frame\_btn\_reset.grid(

row=0, column=1, padx=10, pady=(10, 10))

# Дописать функции вызова

self.third\_frame\_btn\_save = customtkinter.CTkButton(

self.button\_frame, text="Сохранить", command=self.btn\_save\_1\_event)

self.third\_frame\_btn\_save.grid(

row=0, column=2, padx=10, pady=(10, 10))

# select default frame

self.select\_frame\_by\_name("chanel\_selector")

def btn\_load\_1\_event(self):

try:

self.stick\_sensity = np.load('sensity\_settings.npy')

self.third\_frame\_btn\_label\_1 = customtkinter.CTkLabel(

self.button\_frame, text='Файл загружен!', fg\_color="transparent")

self.third\_frame\_btn\_label\_1.grid(

row=1, column=0, padx=10, pady=10)

self.slider\_1.set(

self.stick\_sensity[self.radiobutton\_var.get(), 0])

self.slider\_2.set(

self.stick\_sensity[self.radiobutton\_var.get(), 1])

self.slider\_3.set(

self.stick\_sensity[self.radiobutton\_var.get(), 2])

self.slider\_4.set(

self.stick\_sensity[self.radiobutton\_var.get(), 3])

self.slider\_5.set(

self.stick\_sensity[self.radiobutton\_var.get(), 4])

self.slider\_6.set(

self.stick\_sensity[self.radiobutton\_var.get(), 5])

self.slider\_7.set(

self.stick\_sensity[self.radiobutton\_var.get(), 6])

self.slider\_8.set(

self.stick\_sensity[self.radiobutton\_var.get(), 7])

self.slider\_9.set(

self.stick\_sensity[self.radiobutton\_var.get(), 8])

self.aprox\_sensing()

except:

self.third\_frame\_btn\_label\_1 = customtkinter.CTkLabel(

self.button\_frame, text='Файл не найден!', fg\_color="transparent")

self.third\_frame\_btn\_label\_1.grid(

row=1, column=0, padx=10, pady=10)

def btn\_reset\_1\_event(self):

self.stick\_sensity[self.radiobutton\_var.get(), :] = 1.

self.slider\_1.set(self.stick\_sensity[self.radiobutton\_var.get(), 0])

self.slider\_2.set(self.stick\_sensity[self.radiobutton\_var.get(), 1])

self.slider\_3.set(self.stick\_sensity[self.radiobutton\_var.get(), 2])

self.slider\_4.set(self.stick\_sensity[self.radiobutton\_var.get(), 3])

self.slider\_5.set(self.stick\_sensity[self.radiobutton\_var.get(), 4])

self.slider\_6.set(self.stick\_sensity[self.radiobutton\_var.get(), 5])

self.slider\_7.set(self.stick\_sensity[self.radiobutton\_var.get(), 6])

self.slider\_8.set(self.stick\_sensity[self.radiobutton\_var.get(), 7])

self.slider\_9.set(self.stick\_sensity[self.radiobutton\_var.get(), 8])

self.third\_frame\_btn\_label\_3 = customtkinter.CTkLabel(

self.button\_frame, text='Настройки текущего \n канала сброшены!', fg\_color="transparent")

self.third\_frame\_btn\_label\_3.grid(

row=1, column=1, padx=10, pady=10)

self.aprox\_sensing()

def btn\_save\_1\_event(self):

np.save('sensity\_settings', self.stick\_sensity)

self.third\_frame\_btn\_label\_2 = customtkinter.CTkLabel(

self.button\_frame, text='Файл сохранен!', fg\_color="transparent")

self.third\_frame\_btn\_label\_2.grid(row=1, column=2)

self.aprox\_sensing()

def radiobutton\_event(self):

self.slider\_1.set(self.stick\_sensity[self.radiobutton\_var.get(), 0])

self.slider\_2.set(self.stick\_sensity[self.radiobutton\_var.get(), 1])

self.slider\_3.set(self.stick\_sensity[self.radiobutton\_var.get(), 2])

self.slider\_4.set(self.stick\_sensity[self.radiobutton\_var.get(), 3])

self.slider\_5.set(self.stick\_sensity[self.radiobutton\_var.get(), 4])

self.slider\_6.set(self.stick\_sensity[self.radiobutton\_var.get(), 5])

self.slider\_7.set(self.stick\_sensity[self.radiobutton\_var.get(), 6])

self.slider\_8.set(self.stick\_sensity[self.radiobutton\_var.get(), 7])

self.slider\_9.set(self.stick\_sensity[self.radiobutton\_var.get(), 8])

self.aprox\_sensing()

print(self.radiobutton\_var.get())

def aprox\_sensing(self):

x = list(range(self.stick\_sensity.shape[1]))

x = np.array(x)

x = x / np.max(x)

for i in range(self.stick\_sensity.shape[0]):

y = self.stick\_sensity[i, :]

model = np.polyfit(x, y, 9)

self.aprox\_models[i, :] = model

def set\_sensing\_settings(self, chanel, data):

data = data / 255

predict = np.poly1d(self.aprox\_models[chanel, :])

data = predict(data)

print()

if data > 1:

data = 1

if data < 0:

data = 0

return data

def select\_frame\_by\_name(self, name):

# set button color for selected button

self.joystik\_button.configure(

fg\_color=("gray75", "gray25") if name == "joystik" else "transparent")

self.chanel\_selector\_button.configure(fg\_color=(

"gray75", "gray25") if name == "chanel\_selector" else "transparent")

self.settings\_button.configure(

fg\_color=("gray75", "gray25") if name == "settings" else "transparent")

# show selected frame

if name == "joystik":

self.joystik\_frame.grid(row=0, column=1, sticky="nsew")

self.combobox\_com.configure(values=self.scan\_com())

else:

self.joystik\_frame.grid\_forget()

if name == "chanel\_selector":

self.second\_frame.grid(row=0, column=1, sticky="nsew")

else:

self.second\_frame.grid\_forget()

if name == "settings":

self.radiobutton\_1.configure(

text='Канал 1 ' + self.channel\_list[0])

self.radiobutton\_2.configure(

text='Канал 2 ' + self.channel\_list[1])

self.radiobutton\_3.configure(

text='Канал 3 ' + self.channel\_list[2])

self.radiobutton\_4.configure(

text='Канал 4 ' + self.channel\_list[3])

self.radiobutton\_5.configure(

text='Канал 5 ' + self.channel\_list[4])

self.radiobutton\_6.configure(

text='Канал 6 ' + self.channel\_list[5])

self.radiobutton\_7.configure(

text='Канал 7 ' + self.channel\_list[6])

self.radiobutton\_8.configure(

text='Канал 8 ' + self.channel\_list[7])

self.third\_frame.grid(row=0, column=1, sticky="nsew")

self.aprox\_sensing()

else:

self.third\_frame.grid\_forget()

def joystik\_button\_event(self):

self.select\_frame\_by\_name("joystik")

def chanel\_selector\_button\_event(self):

self.select\_frame\_by\_name("chanel\_selector")

def settings\_button\_event(self):

self.select\_frame\_by\_name("settings")

def button\_start\_event(self):

self.aprox\_sensing()

# create textbox

self.textbox = customtkinter.CTkTextbox(self.joystik\_frame)

self.textbox.grid(row=1, column=0, padx=(

10, 10), pady=(20, 10), sticky="nsew")

pads = inputs\_lib.devices.gamepads

if self.com\_port == '':

self.textbox.insert(

"0.0", "Произведите выбор СОМ порта для \nотправки команд.\n\n")

else:

try:

self.trig = 0

self.thread.join() # останавливаем предыдущий сеанс если он запущен

time.sleep(1)

self.trig = 1

self.thread = Thread(target=self.threading\_event)

self.thread.start()

thread\_name = self.thread.name

self.textbox.insert(

"0.0", "Команды отправляются в " + self.com\_port + '\n\n')

self.textbox.insert(

"0.0", f'Сеанс управления №: {thread\_name}' + '\n\n')

self.textbox.insert(

"0.0", "Запущено ручное управление!\n\n" + "Джойстик: " + str(pads[0]) + '\n\n')

except:

self.trig = 1

self.thread = Thread(target=self.threading\_event)

self.thread.start()

thread\_name = self.thread.name

self.textbox.insert(

"0.0", "Команды отправляются в " + self.com\_port + '\n\n')

self.textbox.insert(

"0.0", f'Сеанс управления №: {thread\_name}' + '\n\n')

self.textbox.insert(

"0.0", "Запущено ручное управление!\n\n" + "Джойстик: " + str(pads[0]) + '\n\n')

def button\_stop\_event(self):

self.trig = 0

self.textbox = customtkinter.CTkTextbox(self.joystik\_frame)

self.textbox.grid(row=1, column=0, padx=(

10, 10), pady=(20, 10), sticky="nsew")

try:

self.thread.join()

self.textbox.insert(

"0.0", "Ручное управление остановлено!\n\n")

except:

self.textbox.insert(

"0.0", "Не найдено сеансов управления!\n\n")

def scan\_com(self):

list\_com = []

for i in range(256):

try:

name = 'COM' + str(i)

s = serial.Serial(name)

list\_com.append(s.name)

s.close() # explicit close 'cause of delayed GC in java

except serial.SerialException:

pass

return list\_com

def combobox\_com\_event(self, choice):

self.com\_port = self.combobox\_com.get()

if self.com\_port == '':

self.textbox.insert(

"0.0", "Произведите выбор СОМ порта для \nотправки команд.\n\n")

else:

self.textbox.insert(

"0.0", 'Выбран ' + self.com\_port + "\nНажмите старт для обнаружения \nджойстика и запуска потока управления.\n\n")

print(self.com\_port)

# Настройка каналов

def clear\_events(self):

while True:

try:

events = inputs\_lib.get\_gamepad(blocking=False)

except inputs\_lib.UnpluggedError:

time.sleep(1)

break

except inputs\_lib.NoDataError:

break

def btn\_ch\_1\_event(self):

self.trig = 0

try:

self.thread.join()

self.clear\_events()

events = inputs\_lib.get\_gamepad(blocking=True)

for event in events:

self.channel\_list[0] = event.code

self.second\_frame\_btn\_ch\_1.configure(

text='Канал 1 ' + event.code)

self.clear\_events()

except:

self.clear\_events()

events = inputs\_lib.get\_gamepad(blocking=True)

for event in events:

self.channel\_list[0] = event.code

self.second\_frame\_btn\_ch\_1.configure(

text='Канал 1 ' + event.code)

self.clear\_events()

def btn\_ch\_2\_event(self):

self.trig = 0

try:

self.thread.join()

self.clear\_events()

events = inputs\_lib.get\_gamepad(blocking=True)

for event in events:

self.channel\_list[1] = event.code

self.second\_frame\_btn\_ch\_2.configure(

text='Канал 2 ' + event.code)

self.clear\_events()

except:

self.clear\_events()

events = inputs\_lib.get\_gamepad(blocking=True)

for event in events:

self.channel\_list[1] = event.code

self.second\_frame\_btn\_ch\_2.configure(

text='Канал 2 ' + event.code)

self.clear\_events()

def btn\_ch\_3\_event(self):

self.trig = 0

try:

self.thread.join()

self.clear\_events()

events = inputs\_lib.get\_gamepad(blocking=True)

for event in events:

self.channel\_list[2] = event.code

self.second\_frame\_btn\_ch\_3.configure(

text='Канал 3 ' + event.code)

self.clear\_events()

except:

self.clear\_events()

events = inputs\_lib.get\_gamepad(blocking=True)

for event in events:

self.channel\_list[2] = event.code

self.second\_frame\_btn\_ch\_3.configure(

text='Канал 3 ' + event.code)

self.clear\_events()

def btn\_ch\_4\_event(self):

self.trig = 0

try:

self.thread.join()

self.clear\_events()

events = inputs\_lib.get\_gamepad(blocking=True)

for event in events:

self.channel\_list[3] = event.code

self.second\_frame\_btn\_ch\_4.configure(

text='Канал 4 ' + event.code)

self.clear\_events()

except:

self.clear\_events()

events = inputs\_lib.get\_gamepad(blocking=True)

for event in events:

self.channel\_list[3] = event.code

self.second\_frame\_btn\_ch\_4.configure(

text='Канал 4 ' + event.code)

self.clear\_events()

def btn\_ch\_5\_event(self):

self.trig = 0

try:

self.thread.join()

self.clear\_events()

events = inputs\_lib.get\_gamepad(blocking=True)

for event in events:

self.channel\_list[4] = event.code

self.second\_frame\_btn\_ch\_5.configure(

text='Канал 5 ' + event.code)

self.clear\_events()

except:

self.clear\_events()

events = inputs\_lib.get\_gamepad(blocking=True)

for event in events:

self.channel\_list[4] = event.code

self.second\_frame\_btn\_ch\_5.configure(

text='Канал 5 ' + event.code)

self.clear\_events()

def btn\_ch\_6\_event(self):

self.trig = 0

try:

self.thread.join()

self.clear\_events()

events = inputs\_lib.get\_gamepad(blocking=True)

for event in events:

self.channel\_list[5] = event.code

self.second\_frame\_btn\_ch\_6.configure(

text='Канал 6 ' + event.code)

self.clear\_events()

except:

self.clear\_events()

events = inputs\_lib.get\_gamepad(blocking=True)

for event in events:

self.channel\_list[5] = event.code

self.second\_frame\_btn\_ch\_6.configure(

text='Канал 6 ' + event.code)

self.clear\_events()

def btn\_ch\_7\_event(self):

self.trig = 0

try:

self.thread.join()

self.clear\_events()

events = inputs\_lib.get\_gamepad(blocking=True)

for event in events:

self.channel\_list[6] = event.code

self.second\_frame\_btn\_ch\_7.configure(

text='Канал 7 ' + event.code)

self.clear\_events()

except:

self.clear\_events()

events = inputs\_lib.get\_gamepad(blocking=True)

for event in events:

self.channel\_list[6] = event.code

self.second\_frame\_btn\_ch\_7.configure(

text='Канал 7 ' + event.code)

self.clear\_events()

def btn\_ch\_8\_event(self):

self.trig = 0

try:

self.thread.join()

self.clear\_events()

events = inputs\_lib.get\_gamepad(blocking=True)

for event in events:

self.channel\_list[7] = event.code

self.second\_frame\_btn\_ch\_8.configure(

text='Канал 8 ' + event.code)

self.clear\_events()

except:

self.clear\_events()

events = inputs\_lib.get\_gamepad(blocking=True)

for event in events:

self.channel\_list[7] = event.code

self.second\_frame\_btn\_ch\_8.configure(

text='Канал 8 ' + event.code)

self.clear\_events()

# Сохранение настроек в файл

def btn\_save\_event(self):

channel\_settings = np.array(self.channel\_list)

np.save('channel\_settings', channel\_settings)

self.second\_frame\_label\_2 = customtkinter.CTkLabel(

self.second\_frame, text='Файл сохранен!', width=120, height=25, fg\_color="transparent", corner\_radius=0)

self.second\_frame\_label\_2.grid(row=2, column=2, padx=10, pady=10)

# Сброс настроек

def btn\_reset\_event(self):

self.channel\_list = ['', '', '', '', '', '', '', '']

self.second\_frame\_btn\_ch\_1.configure(

text='Канал 1 ' + str(self.channel\_list[0]))

self.second\_frame\_btn\_ch\_2.configure(

text='Канал 2 ' + str(self.channel\_list[1]))

self.second\_frame\_btn\_ch\_3.configure(

text='Канал 3 ' + str(self.channel\_list[2]))

self.second\_frame\_btn\_ch\_4.configure(

text='Канал 4 ' + str(self.channel\_list[3]))

self.second\_frame\_btn\_ch\_5.configure(

text='Канал 5 ' + str(self.channel\_list[4]))

self.second\_frame\_btn\_ch\_6.configure(

text='Канал 6 ' + str(self.channel\_list[5]))

self.second\_frame\_btn\_ch\_7.configure(

text='Канал 7 ' + str(self.channel\_list[6]))

self.second\_frame\_btn\_ch\_8.configure(

text='Канал 8 ' + str(self.channel\_list[7]))

self.second\_frame\_label\_2 = customtkinter.CTkLabel(

self.second\_frame, text='Настройки сброшены!', width=120, height=25, fg\_color="transparent", corner\_radius=0)

self.second\_frame\_label\_2.grid(row=4, column=2, padx=10, pady=10)

def btn\_download\_event(self):

try:

channel\_settings = np.load('channel\_settings.npy')

self.channel\_list = list(channel\_settings)

self.second\_frame\_label\_2 = customtkinter.CTkLabel(

self.second\_frame, text='Файл загружен!', width=120, height=25, fg\_color="transparent", corner\_radius=0)

self.second\_frame\_label\_2.grid(row=2, column=0, padx=10, pady=10)

self.second\_frame\_btn\_ch\_1.configure(

text='Канал 1 ' + str(self.channel\_list[0]))

self.second\_frame\_btn\_ch\_2.configure(

text='Канал 2 ' + str(self.channel\_list[1]))

self.second\_frame\_btn\_ch\_3.configure(

text='Канал 3 ' + str(self.channel\_list[2]))

self.second\_frame\_btn\_ch\_4.configure(

text='Канал 4 ' + str(self.channel\_list[3]))

self.second\_frame\_btn\_ch\_5.configure(

text='Канал 5 ' + str(self.channel\_list[4]))

self.second\_frame\_btn\_ch\_6.configure(

text='Канал 6 ' + str(self.channel\_list[5]))

self.second\_frame\_btn\_ch\_7.configure(

text='Канал 7 ' + str(self.channel\_list[6]))

self.second\_frame\_btn\_ch\_8.configure(

text='Канал 8 ' + str(self.channel\_list[7]))

except:

self.second\_frame\_label\_2 = customtkinter.CTkLabel(

self.second\_frame, text='Файл не найден!', width=120, height=25, fg\_color="transparent", corner\_radius=0)

self.second\_frame\_label\_2.grid(row=2, column=0, padx=10, pady=10)

def slider\_1\_callback(self, value):

self.slider\_1.set(value)

self.stick\_sensity[self.radiobutton\_var.get(),

0] = self.slider\_1.get()

print(self.stick\_sensity[self.radiobutton\_var.get(), 0])

print(self.radiobutton\_var.get())

self.aprox\_sensing()

def slider\_2\_callback(self, value):

self.slider\_2.set(value)

self.stick\_sensity[self.radiobutton\_var.get(),

1] = self.slider\_2.get()

print(self.stick\_sensity[self.radiobutton\_var.get(), 1])

self.aprox\_sensing()

def slider\_3\_callback(self, value):

self.slider\_3.set(value)

self.stick\_sensity[self.radiobutton\_var.get(),

2] = self.slider\_3.get()

print(self.stick\_sensity[self.radiobutton\_var.get(), 2])

self.aprox\_sensing()

def slider\_4\_callback(self, value):

self.slider\_4.set(value)

self.stick\_sensity[self.radiobutton\_var.get(),

3] = self.slider\_4.get()

print(self.stick\_sensity[self.radiobutton\_var.get(), 3])

self.aprox\_sensing()

def slider\_5\_callback(self, value):

self.slider\_5.set(value)

self.stick\_sensity[self.radiobutton\_var.get(),

4] = self.slider\_5.get()

print(self.stick\_sensity[self.radiobutton\_var.get(), 4])

self.aprox\_sensing()

def slider\_6\_callback(self, value):

self.slider\_6.set(value)

self.stick\_sensity[self.radiobutton\_var.get(),

5] = self.slider\_6.get()

print(self.stick\_sensity[self.radiobutton\_var.get(), 5])

self.aprox\_sensing()

def slider\_7\_callback(self, value):

self.slider\_7.set(value)

self.stick\_sensity[self.radiobutton\_var.get(),

6] = self.slider\_7.get()

print(self.stick\_sensity[self.radiobutton\_var.get(), 6])

self.aprox\_sensing()

def slider\_8\_callback(self, value):

self.slider\_8.set(value)

self.stick\_sensity[self.radiobutton\_var.get(),

7] = self.slider\_8.get()

print(self.stick\_sensity[self.radiobutton\_var.get(), 7])

self.aprox\_sensing()

def slider\_9\_callback(self, value):

self.slider\_9.set(value)

self.stick\_sensity[self.radiobutton\_var.get(),

8] = self.slider\_9.get()

print(self.stick\_sensity[self.radiobutton\_var.get(), 8])

self.aprox\_sensing()

# Чтение с джойстика и отправка в сериал порт

def threading\_event(self):

if self.WIN:

serial\_port = serial.Serial(

port=self.com\_port, baudrate=9600, timeout=0.1)

serial\_port.isOpen()

while True:

if self.trig == 0:

serial\_port.close()

break

try:

events = inputs\_lib.get\_gamepad(blocking=False)

for event in events:

if event.ev\_type == 'Absolute' and (event.code in self.channel\_list) == True:

#print('event.ev\_type', event.ev\_type)

#print('event.code', event.code)

if self.WIN:

data = ctypes.c\_ubyte(

ctypes.c\_short(event.state).value).value

if self.NIX:

data = event.state

chanel = self.channel\_list.index(event.code)

data = self.set\_sensing\_settings(chanel, data)

mess = ('$' + str(data) + ',' +

str(data) + ',' + str(data) + ';')

serial\_port.write(mess.encode())

print('ch: ', chanel, ', value: ', data)

if event.ev\_type == 'Key' and (event.code in self.channel\_list) == True:

mess = ('$'+str(data)+','+str(data)+','+str(data)+';')

data = event.state

serial\_port.write(mess.encode())

print('ch: ', chanel, ', value: ', data)

except inputs\_lib.UnpluggedError:

self.textbox.insert("0.0", "Джойстик не подключен!\n\n")

time.sleep(1)

continue

except inputs\_lib.NoDataError:

continue

if \_\_name\_\_ == "\_\_main\_\_":

app = App()

app.mainloop()

# Содержимое библиотеки «inputs\_lib»

"""Inputs - user input for humans.

Inputs aims to provide easy to use, cross-platform, user input device

support for Python. I.e. keyboards, mice, gamepads, etc.

Currently supported platforms are the Raspberry Pi, Linux, Windows and

Mac OS X.

"""

# Copyright (c) 2016, 2018: Zeth

# All rights reserved.

#

# BSD Licence

#

# Redistribution and use in source and binary forms, with or without

# modification, are permitted provided that the following conditions are met:

#

# \* Redistributions of source code must retain the above copyright

# notice, this list of conditions and the following disclaimer.

#

# \* Redistributions in binary form must reproduce the above copyright

# notice, this list of conditions and the following disclaimer in the

# documentation and/or other materials provided with the distribution.

#

# \* Neither the name of the copyright holder nor the

# names of its contributors may be used to endorse or promote products

# derived from this software without specific prior written permission.

#

# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"

# AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE

# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE

# ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE

# LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR

# CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF

# SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS

# INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN

# CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)

# ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE

# POSSIBILITY OF SUCH DAMAGE.

from \_\_future\_\_ import print\_function

from \_\_future\_\_ import division

import os

import sys

import io

import glob

import struct

import platform

import math

import time

import codecs

from warnings import warn

from itertools import count

from operator import itemgetter

from multiprocessing import Process, Pipe

import ctypes

\_\_version\_\_ = "0.5"

WIN = True if platform.system() == 'Windows' else False

MAC = True if platform.system() == 'Darwin' else False

NIX = True if platform.system() == 'Linux' else False

if WIN:

# pylint: disable=wrong-import-position

import ctypes.wintypes

DWORD = ctypes.wintypes.DWORD

HANDLE = ctypes.wintypes.HANDLE

WPARAM = ctypes.wintypes.WPARAM

LPARAM = ctypes.wintypes.WPARAM

MSG = ctypes.wintypes.MSG

else:

DWORD = ctypes.c\_ulong

HANDLE = ctypes.c\_void\_p

WPARAM = ctypes.c\_ulonglong

LPARAM = ctypes.c\_ulonglong

MSG = ctypes.Structure

if NIX:

from fcntl import ioctl

OLD = sys.version\_info < (3, 4)

PERMISSIONS\_ERROR\_TEXT = (

"The user (that this program is being run as) does "

"not have permission to access the input events, "

"check groups and permissions, for example, on "

"Debian, the user needs to be in the input group.")

# Standard event format for most devices.

# long, long, unsigned short, unsigned short, int

EVENT\_FORMAT = str('llHHi')

EVENT\_SIZE = struct.calcsize(EVENT\_FORMAT)

def chunks(raw):

"""Yield successive EVENT\_SIZE sized chunks from raw."""

for i in range(0, len(raw), EVENT\_SIZE):

yield struct.unpack(EVENT\_FORMAT, raw[i:i+EVENT\_SIZE])

if OLD:

def iter\_unpack(raw):

"""Yield successive EVENT\_SIZE chunks from message."""

return chunks(raw)

else:

def iter\_unpack(raw):

"""Yield successive EVENT\_SIZE chunks from message."""

return struct.iter\_unpack(EVENT\_FORMAT, raw)

def convert\_timeval(seconds\_since\_epoch):

"""Convert time into C style timeval."""

frac, whole = math.modf(seconds\_since\_epoch)

microseconds = math.floor(frac \* 1000000)

seconds = math.floor(whole)

return seconds, microseconds

SPECIAL\_DEVICES = (

("Raspberry Pi Sense HAT Joystick",

"/dev/input/by-id/gpio-Raspberry\_Pi\_Sense\_HAT\_Joystick-event-kbd"),

("Nintendo Wii Remote",

"/dev/input/by-id/bluetooth-Nintendo\_Wii\_Remote-event-joystick"),

("FT5406 memory based driver",

"/dev/input/by-id/gpio-Raspberry\_Pi\_Touchscreen\_Display-event-mouse"),

)

XINPUT\_MAPPING = (

(1, 0x11),

(2, 0x11),

(3, 0x10),

(4, 0x10),

(5, 0x13a),

(6, 0x13b),

(7, 0x13d),

(8, 0x13e),

(9, 0x136),

(10, 0x137),

(13, 0x130),

(14, 0x131),

(15, 0x134),

(16, 0x133),

(17, 0x11),

('l\_thumb\_x', 0x00),

('l\_thumb\_y', 0x01),

('left\_trigger', 0x02),

('r\_thumb\_x', 0x03),

('r\_thumb\_y', 0x04),

('right\_trigger', 0x05),

)

XINPUT\_DLL\_NAMES = (

"XInput1\_4.dll",

"XInput9\_1\_0.dll",

"XInput1\_3.dll",

"XInput1\_2.dll",

"XInput1\_1.dll"

)

XINPUT\_ERROR\_DEVICE\_NOT\_CONNECTED = 1167

XINPUT\_ERROR\_SUCCESS = 0

XBOX\_STYLE\_LED\_CONTROL = {

0: 'off',

1: 'all blink, then previous setting',

2: '1/top-left blink, then on',

3: '2/top-right blink, then on',

4: '3/bottom-left blink, then on',

5: '4/bottom-right blink, then on',

6: '1/top-left on',

7: '2/top-right on',

8: '3/bottom-left on',

9: '4/bottom-right on',

10: 'rotate',

11: 'blink, based on previous setting',

12: 'slow blink, based on previous setting',

13: 'rotate with two lights',

14: 'persistent slow all blink',

15: 'blink once, then previous setting'

}

DEVICE\_PROPERTIES = (

(0x00, "INPUT\_PROP\_POINTER"), # needs a pointer

(0x01, "INPUT\_PROP\_DIRECT"), # direct input devices

(0x02, "INPUT\_PROP\_BUTTONPAD"), # has button(s) under pad

(0x03, "INPUT\_PROP\_SEMI\_MT"), # touch rectangle only

(0x04, "INPUT\_PROP\_TOPBUTTONPAD"), # softbuttons at top of pad

(0x05, "INPUT\_PROP\_POINTING\_STICK"), # is a pointing stick

(0x06, "INPUT\_PROP\_ACCELEROMETER"), # has accelerometer

(0x1f, "INPUT\_PROP\_MAX"),

(0x1f + 1, "INPUT\_PROP\_CNT"))

EVENT\_TYPES = (

(0x00, "Sync"),

(0x01, "Key"),

(0x02, "Relative"),

(0x03, "Absolute"),

(0x04, "Misc"),

(0x05, "Switch"),

(0x11, "LED"),

(0x12, "Sound"),

(0x14, "Repeat"),

(0x15, "ForceFeedback"),

(0x16, "Power"),

(0x17, "ForceFeedbackStatus"),

(0x1f, "Max"),

(0x1f+1, "Current"))

SYNCHRONIZATION\_EVENTS = (

(0, "SYN\_REPORT"),

(1, "SYN\_CONFIG"),

(2, "SYN\_MT\_REPORT"),

(3, "SYN\_DROPPED"),

(0xf, "SYN\_MAX"),

(0xf+1, "SYN\_CNT"))

KEYS\_AND\_BUTTONS = (

(0, "KEY\_RESERVED"),

(1, "KEY\_ESC"),

(2, "KEY\_1"),

(3, "KEY\_2"),

(4, "KEY\_3"),

(5, "KEY\_4"),

(6, "KEY\_5"),

(7, "KEY\_6"),

(8, "KEY\_7"),

(9, "KEY\_8"),

(10, "KEY\_9"),

(11, "KEY\_0"),

(12, "KEY\_MINUS"),

(13, "KEY\_EQUAL"),

(14, "KEY\_BACKSPACE"),

(15, "KEY\_TAB"),

(16, "KEY\_Q"),

(17, "KEY\_W"),

(18, "KEY\_E"),

(19, "KEY\_R"),

(20, "KEY\_T"),

(21, "KEY\_Y"),

(22, "KEY\_U"),

(23, "KEY\_I"),

(24, "KEY\_O"),

(25, "KEY\_P"),

(26, "KEY\_LEFTBRACE"),

(27, "KEY\_RIGHTBRACE"),

(28, "KEY\_ENTER"),

(29, "KEY\_LEFTCTRL"),

(30, "KEY\_A"),

(31, "KEY\_S"),

(32, "KEY\_D"),

(33, "KEY\_F"),

(34, "KEY\_G"),

(35, "KEY\_H"),

(36, "KEY\_J"),

(37, "KEY\_K"),

(38, "KEY\_L"),

(39, "KEY\_SEMICOLON"),

(40, "KEY\_APOSTROPHE"),

(41, "KEY\_GRAVE"),

(42, "KEY\_LEFTSHIFT"),

(43, "KEY\_BACKSLASH"),

(44, "KEY\_Z"),

(45, "KEY\_X"),

(46, "KEY\_C"),

(47, "KEY\_V"),

(48, "KEY\_B"),

(49, "KEY\_N"),

(50, "KEY\_M"),

(51, "KEY\_COMMA"),

(52, "KEY\_DOT"),

(53, "KEY\_SLASH"),

(54, "KEY\_RIGHTSHIFT"),

(55, "KEY\_KPASTERISK"),

(56, "KEY\_LEFTALT"),

(57, "KEY\_SPACE"),

(58, "KEY\_CAPSLOCK"),

(59, "KEY\_F1"),

(60, "KEY\_F2"),

(61, "KEY\_F3"),

(62, "KEY\_F4"),

(63, "KEY\_F5"),

(64, "KEY\_F6"),

(65, "KEY\_F7"),

(66, "KEY\_F8"),

(67, "KEY\_F9"),

(68, "KEY\_F10"),

(69, "KEY\_NUMLOCK"),

(70, "KEY\_SCROLLLOCK"),

(71, "KEY\_KP7"),

(72, "KEY\_KP8"),

(73, "KEY\_KP9"),

(74, "KEY\_KPMINUS"),

(75, "KEY\_KP4"),

(76, "KEY\_KP5"),

(77, "KEY\_KP6"),

(78, "KEY\_KPPLUS"),

(79, "KEY\_KP1"),

(80, "KEY\_KP2"),

(81, "KEY\_KP3"),

(82, "KEY\_KP0"),

(83, "KEY\_KPDOT"),

(85, "KEY\_ZENKAKUHANKAKU"),

(86, "KEY\_102ND"),

(87, "KEY\_F11"),

(88, "KEY\_F12"),

(89, "KEY\_RO"),

(90, "KEY\_KATAKANA"),

(91, "KEY\_HIRAGANA"),

(92, "KEY\_HENKAN"),

(93, "KEY\_KATAKANAHIRAGANA"),

(94, "KEY\_MUHENKAN"),

(95, "KEY\_KPJPCOMMA"),

(96, "KEY\_KPENTER"),

(97, "KEY\_RIGHTCTRL"),

(98, "KEY\_KPSLASH"),

(99, "KEY\_SYSRQ"),

(100, "KEY\_RIGHTALT"),

(101, "KEY\_LINEFEED"),

(102, "KEY\_HOME"),

(103, "KEY\_UP"),

(104, "KEY\_PAGEUP"),

(105, "KEY\_LEFT"),

(106, "KEY\_RIGHT"),

(107, "KEY\_END"),

(108, "KEY\_DOWN"),

(109, "KEY\_PAGEDOWN"),

(110, "KEY\_INSERT"),

(111, "KEY\_DELETE"),

(112, "KEY\_MACRO"),

(113, "KEY\_MUTE"),

(114, "KEY\_VOLUMEDOWN"),

(115, "KEY\_VOLUMEUP"),

(116, "KEY\_POWER"), # SC System Power Down

(117, "KEY\_KPEQUAL"),

(118, "KEY\_KPPLUSMINUS"),

(119, "KEY\_PAUSE"),

(120, "KEY\_SCALE"), # AL Compiz Scale (Expose)

(121, "KEY\_KPCOMMA"),

(122, "KEY\_HANGEUL"),

(123, "KEY\_HANJA"),

(124, "KEY\_YEN"),

(125, "KEY\_LEFTMETA"),

(126, "KEY\_RIGHTMETA"),

(127, "KEY\_COMPOSE"),

(128, "KEY\_STOP"), # AC Stop

(129, "KEY\_AGAIN"),

(130, "KEY\_PROPS"), # AC Properties

(131, "KEY\_UNDO"), # AC Undo

(132, "KEY\_FRONT"),

(133, "KEY\_COPY"), # AC Copy

(134, "KEY\_OPEN"), # AC Open

(135, "KEY\_PASTE"), # AC Paste

(136, "KEY\_FIND"), # AC Search

(137, "KEY\_CUT"), # AC Cut

(138, "KEY\_HELP"), # AL Integrated Help Center

(139, "KEY\_MENU"), # Menu (show menu)

(140, "KEY\_CALC"), # AL Calculator

(141, "KEY\_SETUP"),

(142, "KEY\_SLEEP"), # SC System Sleep

(143, "KEY\_WAKEUP"), # System Wake Up

(144, "KEY\_FILE"), # AL Local Machine Browser

(145, "KEY\_SENDFILE"),

(146, "KEY\_DELETEFILE"),

(147, "KEY\_XFER"),

(148, "KEY\_PROG1"),

(149, "KEY\_PROG2"),

(150, "KEY\_WWW"), # AL Internet Browser

(151, "KEY\_MSDOS"),

(152, "KEY\_COFFEE"), # AL Terminal Lock/Screensaver

(153, "KEY\_ROTATE\_DISPLAY"), # Display orientation for e.g. tablets

(154, "KEY\_CYCLEWINDOWS"),

(155, "KEY\_MAIL"),

(156, "KEY\_BOOKMARKS"), # AC Bookmarks

(157, "KEY\_COMPUTER"),

(158, "KEY\_BACK"), # AC Back

(159, "KEY\_FORWARD"), # AC Forward

(160, "KEY\_CLOSECD"),

(161, "KEY\_EJECTCD"),

(162, "KEY\_EJECTCLOSECD"),

(163, "KEY\_NEXTSONG"),

(164, "KEY\_PLAYPAUSE"),

(165, "KEY\_PREVIOUSSONG"),

(166, "KEY\_STOPCD"),

(167, "KEY\_RECORD"),

(168, "KEY\_REWIND"),

(169, "KEY\_PHONE"), # Media Select Telephone

(170, "KEY\_ISO"),

(171, "KEY\_CONFIG"), # AL Consumer Control Configuration

(172, "KEY\_HOMEPAGE"), # AC Home

(173, "KEY\_REFRESH"), # AC Refresh

(174, "KEY\_EXIT"), # AC Exit

(175, "KEY\_MOVE"),

(176, "KEY\_EDIT"),

(177, "KEY\_SCROLLUP"),

(178, "KEY\_SCROLLDOWN"),

(179, "KEY\_KPLEFTPAREN"),

(180, "KEY\_KPRIGHTPAREN"),

(181, "KEY\_NEW"), # AC New

(182, "KEY\_REDO"), # AC Redo/Repeat

(183, "KEY\_F13"),

(184, "KEY\_F14"),

(185, "KEY\_F15"),

(186, "KEY\_F16"),

(187, "KEY\_F17"),

(188, "KEY\_F18"),

(189, "KEY\_F19"),

(190, "KEY\_F20"),

(191, "KEY\_F21"),

(192, "KEY\_F22"),

(193, "KEY\_F23"),

(194, "KEY\_F24"),

(200, "KEY\_PLAYCD"),

(201, "KEY\_PAUSECD"),

(202, "KEY\_PROG3"),

(203, "KEY\_PROG4"),

(204, "KEY\_DASHBOARD"), # AL Dashboard

(205, "KEY\_SUSPEND"),

(206, "KEY\_CLOSE"), # AC Close

(207, "KEY\_PLAY"),

(208, "KEY\_FASTFORWARD"),

(209, "KEY\_BASSBOOST"),

(210, "KEY\_PRINT"), # AC Print

(211, "KEY\_HP"),

(212, "KEY\_CAMERA"),

(213, "KEY\_SOUND"),

(214, "KEY\_QUESTION"),

(215, "KEY\_EMAIL"),

(216, "KEY\_CHAT"),

(217, "KEY\_SEARCH"),

(218, "KEY\_CONNECT"),

(219, "KEY\_FINANCE"), # AL Checkbook/Finance

(220, "KEY\_SPORT"),

(221, "KEY\_SHOP"),

(222, "KEY\_ALTERASE"),

(223, "KEY\_CANCEL"), # AC Cancel

(224, "KEY\_BRIGHTNESSDOWN"),

(225, "KEY\_BRIGHTNESSUP"),

(226, "KEY\_MEDIA"),

(227, "KEY\_SWITCHVIDEOMODE"), # Cycle between available video

(228, "KEY\_KBDILLUMTOGGLE"),

(229, "KEY\_KBDILLUMDOWN"),

(230, "KEY\_KBDILLUMUP"),

(231, "KEY\_SEND"), # AC Send

(232, "KEY\_REPLY"), # AC Reply

(233, "KEY\_FORWARDMAIL"), # AC Forward Msg

(234, "KEY\_SAVE"), # AC Save

(235, "KEY\_DOCUMENTS"),

(236, "KEY\_BATTERY"),

(237, "KEY\_BLUETOOTH"),

(238, "KEY\_WLAN"),

(239, "KEY\_UWB"),

(240, "KEY\_UNKNOWN"),

(241, "KEY\_VIDEO\_NEXT"), # drive next video source

(242, "KEY\_VIDEO\_PREV"), # drive previous video source

(243, "KEY\_BRIGHTNESS\_CYCLE"), # brightness up, after max is min

(244, "KEY\_BRIGHTNESS\_AUTO"), # Set Auto Brightness: manual

(245, "KEY\_DISPLAY\_OFF"), # display device to off state

(246, "KEY\_WWAN"), # Wireless WAN (LTE, UMTS, GSM, etc.)

(247, "KEY\_RFKILL"), # Key that controls all radios

(248, "KEY\_MICMUTE"), # Mute / unmute the microphone

(0x100, "BTN\_MISC"),

(0x100, "BTN\_0"),

(0x101, "BTN\_1"),

(0x102, "BTN\_2"),

(0x103, "BTN\_3"),

(0x104, "BTN\_4"),

(0x105, "BTN\_5"),

(0x106, "BTN\_6"),

(0x107, "BTN\_7"),

(0x108, "BTN\_8"),

(0x109, "BTN\_9"),

(0x110, "BTN\_MOUSE"),

(0x110, "BTN\_LEFT"),

(0x111, "BTN\_RIGHT"),

(0x112, "BTN\_MIDDLE"),

(0x113, "BTN\_SIDE"),

(0x114, "BTN\_EXTRA"),

(0x115, "BTN\_FORWARD"),

(0x116, "BTN\_BACK"),

(0x117, "BTN\_TASK"),

(0x120, "BTN\_JOYSTICK"),

(0x120, "BTN\_TRIGGER"),

(0x121, "BTN\_THUMB"),

(0x122, "BTN\_THUMB2"),

(0x123, "BTN\_TOP"),

(0x124, "BTN\_TOP2"),

(0x125, "BTN\_PINKIE"),

(0x126, "BTN\_BASE"),

(0x127, "BTN\_BASE2"),

(0x128, "BTN\_BASE3"),

(0x129, "BTN\_BASE4"),

(0x12a, "BTN\_BASE5"),

(0x12b, "BTN\_BASE6"),

(0x12f, "BTN\_DEAD"),

(0x130, "BTN\_GAMEPAD"),

(0x130, "BTN\_SOUTH"),

(0x131, "BTN\_EAST"),

(0x132, "BTN\_C"),

(0x133, "BTN\_NORTH"),

(0x134, "BTN\_WEST"),

(0x135, "BTN\_Z"),

(0x136, "BTN\_TL"),

(0x137, "BTN\_TR"),

(0x138, "BTN\_TL2"),

(0x139, "BTN\_TR2"),

(0x13a, "BTN\_SELECT"),

(0x13b, "BTN\_START"),

(0x13c, "BTN\_MODE"),

(0x13d, "BTN\_THUMBL"),

(0x13e, "BTN\_THUMBR"),

(0x140, "BTN\_DIGI"),

(0x140, "BTN\_TOOL\_PEN"),

(0x141, "BTN\_TOOL\_RUBBER"),

(0x142, "BTN\_TOOL\_BRUSH"),

(0x143, "BTN\_TOOL\_PENCIL"),

(0x144, "BTN\_TOOL\_AIRBRUSH"),

(0x145, "BTN\_TOOL\_FINGER"),

(0x146, "BTN\_TOOL\_MOUSE"),

(0x147, "BTN\_TOOL\_LENS"),

(0x148, "BTN\_TOOL\_QUINTTAP"), # Five fingers on trackpad

(0x14a, "BTN\_TOUCH"),

(0x14b, "BTN\_STYLUS"),

(0x14c, "BTN\_STYLUS2"),

(0x14d, "BTN\_TOOL\_DOUBLETAP"),

(0x14e, "BTN\_TOOL\_TRIPLETAP"),

(0x14f, "BTN\_TOOL\_QUADTAP"), # Four fingers on trackpad

(0x150, "BTN\_WHEEL"),

(0x150, "BTN\_GEAR\_DOWN"),

(0x151, "BTN\_GEAR\_UP"),

(0x160, "KEY\_OK"),

(0x161, "KEY\_SELECT"),

(0x162, "KEY\_GOTO"),

(0x163, "KEY\_CLEAR"),

(0x164, "KEY\_POWER2"),

(0x165, "KEY\_OPTION"),

(0x166, "KEY\_INFO"), # AL OEM Features/Tips/Tutorial

(0x167, "KEY\_TIME"),

(0x168, "KEY\_VENDOR"),

(0x169, "KEY\_ARCHIVE"),

(0x16a, "KEY\_PROGRAM"), # Media Select Program Guide

(0x16b, "KEY\_CHANNEL"),

(0x16c, "KEY\_FAVORITES"),

(0x16d, "KEY\_EPG"),

(0x16e, "KEY\_PVR"), # Media Select Home

(0x16f, "KEY\_MHP"),

(0x170, "KEY\_LANGUAGE"),

(0x171, "KEY\_TITLE"),

(0x172, "KEY\_SUBTITLE"),

(0x173, "KEY\_ANGLE"),

(0x174, "KEY\_ZOOM"),

(0x175, "KEY\_MODE"),

(0x176, "KEY\_KEYBOARD"),

(0x177, "KEY\_SCREEN"),

(0x178, "KEY\_PC"), # Media Select Computer

(0x179, "KEY\_TV"), # Media Select TV

(0x17a, "KEY\_TV2"), # Media Select Cable

(0x17b, "KEY\_VCR"), # Media Select VCR

(0x17c, "KEY\_VCR2"), # VCR Plus

(0x17d, "KEY\_SAT"), # Media Select Satellite

(0x17e, "KEY\_SAT2"),

(0x17f, "KEY\_CD"), # Media Select CD

(0x180, "KEY\_TAPE"), # Media Select Tape

(0x181, "KEY\_RADIO"),

(0x182, "KEY\_TUNER"), # Media Select Tuner

(0x183, "KEY\_PLAYER"),

(0x184, "KEY\_TEXT"),

(0x185, "KEY\_DVD"), # Media Select DVD

(0x186, "KEY\_AUX"),

(0x187, "KEY\_MP3"),

(0x188, "KEY\_AUDIO"), # AL Audio Browser

(0x189, "KEY\_VIDEO"), # AL Movie Browser

(0x18a, "KEY\_DIRECTORY"),

(0x18b, "KEY\_LIST"),

(0x18c, "KEY\_MEMO"), # Media Select Messages

(0x18d, "KEY\_CALENDAR"),

(0x18e, "KEY\_RED"),

(0x18f, "KEY\_GREEN"),

(0x190, "KEY\_YELLOW"),

(0x191, "KEY\_BLUE"),

(0x192, "KEY\_CHANNELUP"), # Channel Increment

(0x193, "KEY\_CHANNELDOWN"), # Channel Decrement

(0x194, "KEY\_FIRST"),

(0x195, "KEY\_LAST"), # Recall Last

(0x196, "KEY\_AB"),

(0x197, "KEY\_NEXT"),

(0x198, "KEY\_RESTART"),

(0x199, "KEY\_SLOW"),

(0x19a, "KEY\_SHUFFLE"),

(0x19b, "KEY\_BREAK"),

(0x19c, "KEY\_PREVIOUS"),

(0x19d, "KEY\_DIGITS"),

(0x19e, "KEY\_TEEN"),

(0x19f, "KEY\_TWEN"),

(0x1a0, "KEY\_VIDEOPHONE"), # Media Select Video Phone

(0x1a1, "KEY\_GAMES"), # Media Select Games

(0x1a2, "KEY\_ZOOMIN"), # AC Zoom In

(0x1a3, "KEY\_ZOOMOUT"), # AC Zoom Out

(0x1a4, "KEY\_ZOOMRESET"), # AC Zoom

(0x1a5, "KEY\_WORDPROCESSOR"), # AL Word Processor

(0x1a6, "KEY\_EDITOR"), # AL Text Editor

(0x1a7, "KEY\_SPREADSHEET"), # AL Spreadsheet

(0x1a8, "KEY\_GRAPHICSEDITOR"), # AL Graphics Editor

(0x1a9, "KEY\_PRESENTATION"), # AL Presentation App

(0x1aa, "KEY\_DATABASE"), # AL Database App

(0x1ab, "KEY\_NEWS"), # AL Newsreader

(0x1ac, "KEY\_VOICEMAIL"), # AL Voicemail

(0x1ad, "KEY\_ADDRESSBOOK"), # AL Contacts/Address Book

(0x1ae, "KEY\_MESSENGER"), # AL Instant Messaging

(0x1af, "KEY\_DISPLAYTOGGLE"), # Turn display (LCD) on and off

(0x1b0, "KEY\_SPELLCHECK"), # AL Spell Check

(0x1b1, "KEY\_LOGOFF"), # AL Logoff

(0x1b2, "KEY\_DOLLAR"),

(0x1b3, "KEY\_EURO"),

(0x1b4, "KEY\_FRAMEBACK"), # Consumer - transport controls

(0x1b5, "KEY\_FRAMEFORWARD"),

(0x1b6, "KEY\_CONTEXT\_MENU"), # GenDesc - system context menu

(0x1b7, "KEY\_MEDIA\_REPEAT"), # Consumer - transport control

(0x1b8, "KEY\_10CHANNELSUP"), # 10 channels up (10+)

(0x1b9, "KEY\_10CHANNELSDOWN"), # 10 channels down (10-)

(0x1ba, "KEY\_IMAGES"), # AL Image Browser

(0x1c0, "KEY\_DEL\_EOL"),

(0x1c1, "KEY\_DEL\_EOS"),

(0x1c2, "KEY\_INS\_LINE"),

(0x1c3, "KEY\_DEL\_LINE"),

(0x1d0, "KEY\_FN"),

(0x1d1, "KEY\_FN\_ESC"),

(0x1d2, "KEY\_FN\_F1"),

(0x1d3, "KEY\_FN\_F2"),

(0x1d4, "KEY\_FN\_F3"),

(0x1d5, "KEY\_FN\_F4"),

(0x1d6, "KEY\_FN\_F5"),

(0x1d7, "KEY\_FN\_F6"),

(0x1d8, "KEY\_FN\_F7"),

(0x1d9, "KEY\_FN\_F8"),

(0x1da, "KEY\_FN\_F9"),

(0x1db, "KEY\_FN\_F10"),

(0x1dc, "KEY\_FN\_F11"),

(0x1dd, "KEY\_FN\_F12"),

(0x1de, "KEY\_FN\_1"),

(0x1df, "KEY\_FN\_2"),

(0x1e0, "KEY\_FN\_D"),

(0x1e1, "KEY\_FN\_E"),

(0x1e2, "KEY\_FN\_F"),

(0x1e3, "KEY\_FN\_S"),

(0x1e4, "KEY\_FN\_B"),

(0x1f1, "KEY\_BRL\_DOT1"),

(0x1f2, "KEY\_BRL\_DOT2"),

(0x1f3, "KEY\_BRL\_DOT3"),

(0x1f4, "KEY\_BRL\_DOT4"),

(0x1f5, "KEY\_BRL\_DOT5"),

(0x1f6, "KEY\_BRL\_DOT6"),

(0x1f7, "KEY\_BRL\_DOT7"),

(0x1f8, "KEY\_BRL\_DOT8"),

(0x1f9, "KEY\_BRL\_DOT9"),

(0x1fa, "KEY\_BRL\_DOT10"),

(0x200, "KEY\_NUMERIC\_0"), # used by phones, remote controls,

(0x201, "KEY\_NUMERIC\_1"), # and other keypads

(0x202, "KEY\_NUMERIC\_2"),

(0x203, "KEY\_NUMERIC\_3"),

(0x204, "KEY\_NUMERIC\_4"),

(0x205, "KEY\_NUMERIC\_5"),

(0x206, "KEY\_NUMERIC\_6"),

(0x207, "KEY\_NUMERIC\_7"),

(0x208, "KEY\_NUMERIC\_8"),

(0x209, "KEY\_NUMERIC\_9"),

(0x20a, "KEY\_NUMERIC\_STAR"),

(0x20b, "KEY\_NUMERIC\_POUND"),

(0x20c, "KEY\_NUMERIC\_A"), # Phone key A - HUT Telephony 0xb9

(0x20d, "KEY\_NUMERIC\_B"),

(0x20e, "KEY\_NUMERIC\_C"),

(0x20f, "KEY\_NUMERIC\_D"),

(0x210, "KEY\_CAMERA\_FOCUS"),

(0x211, "KEY\_WPS\_BUTTON"), # WiFi Protected Setup key

(0x212, "KEY\_TOUCHPAD\_TOGGLE"), # Request switch touchpad on or off

(0x213, "KEY\_TOUCHPAD\_ON"),

(0x214, "KEY\_TOUCHPAD\_OFF"),

(0x215, "KEY\_CAMERA\_ZOOMIN"),

(0x216, "KEY\_CAMERA\_ZOOMOUT"),

(0x217, "KEY\_CAMERA\_UP"),

(0x218, "KEY\_CAMERA\_DOWN"),

(0x219, "KEY\_CAMERA\_LEFT"),

(0x21a, "KEY\_CAMERA\_RIGHT"),

(0x21b, "KEY\_ATTENDANT\_ON"),

(0x21c, "KEY\_ATTENDANT\_OFF"),

(0x21d, "KEY\_ATTENDANT\_TOGGLE"), # Attendant call on or off

(0x21e, "KEY\_LIGHTS\_TOGGLE"), # Reading light on or off

(0x220, "BTN\_DPAD\_UP"),

(0x221, "BTN\_DPAD\_DOWN"),

(0x222, "BTN\_DPAD\_LEFT"),

(0x223, "BTN\_DPAD\_RIGHT"),

(0x230, "KEY\_ALS\_TOGGLE"), # Ambient light sensor

(0x240, "KEY\_BUTTONCONFIG"), # AL Button Configuration

(0x241, "KEY\_TASKMANAGER"), # AL Task/Project Manager

(0x242, "KEY\_JOURNAL"), # AL Log/Journal/Timecard

(0x243, "KEY\_CONTROLPANEL"), # AL Control Panel

(0x244, "KEY\_APPSELECT"), # AL Select Task/Application

(0x245, "KEY\_SCREENSAVER"), # AL Screen Saver

(0x246, "KEY\_VOICECOMMAND"), # Listening Voice Command

(0x250, "KEY\_BRIGHTNESS\_MIN"), # Set Brightness to Minimum

(0x251, "KEY\_BRIGHTNESS\_MAX"), # Set Brightness to Maximum

(0x260, "KEY\_KBDINPUTASSIST\_PREV"),

(0x261, "KEY\_KBDINPUTASSIST\_NEXT"),

(0x262, "KEY\_KBDINPUTASSIST\_PREVGROUP"),

(0x263, "KEY\_KBDINPUTASSIST\_NEXTGROUP"),

(0x264, "KEY\_KBDINPUTASSIST\_ACCEPT"),

(0x265, "KEY\_KBDINPUTASSIST\_CANCEL"),

(0x2c0, "BTN\_TRIGGER\_HAPPY"),

(0x2c0, "BTN\_TRIGGER\_HAPPY1"),

(0x2c1, "BTN\_TRIGGER\_HAPPY2"),

(0x2c2, "BTN\_TRIGGER\_HAPPY3"),

(0x2c3, "BTN\_TRIGGER\_HAPPY4"),

(0x2c4, "BTN\_TRIGGER\_HAPPY5"),

(0x2c5, "BTN\_TRIGGER\_HAPPY6"),

(0x2c6, "BTN\_TRIGGER\_HAPPY7"),

(0x2c7, "BTN\_TRIGGER\_HAPPY8"),

(0x2c8, "BTN\_TRIGGER\_HAPPY9"),

(0x2c9, "BTN\_TRIGGER\_HAPPY10"),

(0x2ca, "BTN\_TRIGGER\_HAPPY11"),

(0x2cb, "BTN\_TRIGGER\_HAPPY12"),

(0x2cc, "BTN\_TRIGGER\_HAPPY13"),

(0x2cd, "BTN\_TRIGGER\_HAPPY14"),

(0x2ce, "BTN\_TRIGGER\_HAPPY15"),

(0x2cf, "BTN\_TRIGGER\_HAPPY16"),

(0x2d0, "BTN\_TRIGGER\_HAPPY17"),

(0x2d1, "BTN\_TRIGGER\_HAPPY18"),

(0x2d2, "BTN\_TRIGGER\_HAPPY19"),

(0x2d3, "BTN\_TRIGGER\_HAPPY20"),

(0x2d4, "BTN\_TRIGGER\_HAPPY21"),

(0x2d5, "BTN\_TRIGGER\_HAPPY22"),

(0x2d6, "BTN\_TRIGGER\_HAPPY23"),

(0x2d7, "BTN\_TRIGGER\_HAPPY24"),

(0x2d8, "BTN\_TRIGGER\_HAPPY25"),

(0x2d9, "BTN\_TRIGGER\_HAPPY26"),

(0x2da, "BTN\_TRIGGER\_HAPPY27"),

(0x2db, "BTN\_TRIGGER\_HAPPY28"),

(0x2dc, "BTN\_TRIGGER\_HAPPY29"),

(0x2dd, "BTN\_TRIGGER\_HAPPY30"),

(0x2de, "BTN\_TRIGGER\_HAPPY31"),

(0x2df, "BTN\_TRIGGER\_HAPPY32"),

(0x2e0, "BTN\_TRIGGER\_HAPPY33"),

(0x2e1, "BTN\_TRIGGER\_HAPPY34"),

(0x2e2, "BTN\_TRIGGER\_HAPPY35"),

(0x2e3, "BTN\_TRIGGER\_HAPPY36"),

(0x2e4, "BTN\_TRIGGER\_HAPPY37"),

(0x2e5, "BTN\_TRIGGER\_HAPPY38"),

(0x2e6, "BTN\_TRIGGER\_HAPPY39"),

(0x2e7, "BTN\_TRIGGER\_HAPPY40"),

(0x2ff, "KEY\_MAX"),

(0x2ff+1, "KEY\_CNT"))

RELATIVE\_AXES = (

(0x00, "REL\_X"),

(0x01, "REL\_Y"),

(0x02, "REL\_Z"),

(0x03, "REL\_RX"),

(0x04, "REL\_RY"),

(0x05, "REL\_RZ"),

(0x06, "REL\_HWHEEL"),

(0x07, "REL\_DIAL"),

(0x08, "REL\_WHEEL"),

(0x09, "REL\_MISC"),

(0x0f, "REL\_MAX"),

(0x0f+1, "REL\_CNT"))

ABSOLUTE\_AXES = (

(0x00, "ABS\_X"),

(0x01, "ABS\_Y"),

(0x02, "ABS\_Z"),

(0x03, "ABS\_RX"),

(0x04, "ABS\_RY"),

(0x05, "ABS\_RZ"),

(0x06, "ABS\_THROTTLE"),

(0x07, "ABS\_RUDDER"),

(0x08, "ABS\_WHEEL"),

(0x09, "ABS\_GAS"),

(0x0a, "ABS\_BRAKE"),

(0x10, "ABS\_HAT0X"),

(0x11, "ABS\_HAT0Y"),

(0x12, "ABS\_HAT1X"),

(0x13, "ABS\_HAT1Y"),

(0x14, "ABS\_HAT2X"),

(0x15, "ABS\_HAT2Y"),

(0x16, "ABS\_HAT3X"),

(0x17, "ABS\_HAT3Y"),

(0x18, "ABS\_PRESSURE"),

(0x19, "ABS\_DISTANCE"),

(0x1a, "ABS\_TILT\_X"),

(0x1b, "ABS\_TILT\_Y"),

(0x1c, "ABS\_TOOL\_WIDTH"),

(0x20, "ABS\_VOLUME"),

(0x28, "ABS\_MISC"),

(0x2f, "ABS\_MT\_SLOT"), # MT slot being modified

(0x30, "ABS\_MT\_TOUCH\_MAJOR"), # Major axis of touching ellipse

(0x31, "ABS\_MT\_TOUCH\_MINOR"), # Minor axis (omit if circular)

(0x32, "ABS\_MT\_WIDTH\_MAJOR"), # Major axis of approaching ellipse

(0x33, "ABS\_MT\_WIDTH\_MINOR"), # Minor axis (omit if circular)

(0x34, "ABS\_MT\_ORIENTATION"), # Ellipse orientation

(0x35, "ABS\_MT\_POSITION\_X"), # Center X touch position

(0x36, "ABS\_MT\_POSITION\_Y"), # Center Y touch position

(0x37, "ABS\_MT\_TOOL\_TYPE"), # Type of touching device

(0x38, "ABS\_MT\_BLOB\_ID"), # Group a set of packets as a blob

(0x39, "ABS\_MT\_TRACKING\_ID"), # Unique ID of initiated contact

(0x3a, "ABS\_MT\_PRESSURE"), # Pressure on contact area

(0x3b, "ABS\_MT\_DISTANCE"), # Contact hover distance

(0x3c, "ABS\_MT\_TOOL\_X"), # Center X tool position

(0x3d, "ABS\_MT\_TOOL\_Y"), # Center Y tool position

(0x3f, "ABS\_MAX"),

(0x3f+1, "ABS\_CNT"))

SWITCH\_EVENTS = (

(0x00, "SW\_LID"), # set = lid shut

(0x01, "SW\_TABLET\_MODE"), # set = tablet mode

(0x02, "SW\_HEADPHONE\_INSERT"), # set = inserted

(0x03, "SW\_RFKILL\_ALL"), # rfkill master switch, type "any"

(0x04, "SW\_MICROPHONE\_INSERT"), # set = inserted

(0x05, "SW\_DOCK"), # set = plugged into dock

(0x06, "SW\_LINEOUT\_INSERT"), # set = inserted

(0x07, "SW\_JACK\_PHYSICAL\_INSERT"), # set = mechanical switch set

(0x08, "SW\_VIDEOOUT\_INSERT"), # set = inserted

(0x09, "SW\_CAMERA\_LENS\_COVER"), # set = lens covered

(0x0a, "SW\_KEYPAD\_SLIDE"), # set = keypad slide out

(0x0b, "SW\_FRONT\_PROXIMITY"), # set = front proximity sensor active

(0x0c, "SW\_ROTATE\_LOCK"), # set = rotate locked/disabled

(0x0d, "SW\_LINEIN\_INSERT"), # set = inserted

(0x0e, "SW\_MUTE\_DEVICE"), # set = device disabled

(0x0f, "SW\_MAX"),

(0x0f+1, "SW\_CNT"))

MISC\_EVENTS = (

(0x00, "MSC\_SERIAL"),

(0x01, "MSC\_PULSELED"),

(0x02, "MSC\_GESTURE"),

(0x03, "MSC\_RAW"),

(0x04, "MSC\_SCAN"),

(0x05, "MSC\_TIMESTAMP"),

(0x07, "MSC\_MAX"),

(0x07+1, "MSC\_CNT"))

LEDS = (

(0x00, "LED\_NUML"),

(0x01, "LED\_CAPSL"),

(0x02, "LED\_SCROLLL"),

(0x03, "LED\_COMPOSE"),

(0x04, "LED\_KANA"),

(0x05, "LED\_SLEEP"),

(0x06, "LED\_SUSPEND"),

(0x07, "LED\_MUTE"),

(0x08, "LED\_MISC"),

(0x09, "LED\_MAIL"),

(0x0a, "LED\_CHARGING"),

(0x0f, "LED\_MAX"),

(0x0f+1, "LED\_CNT"))

LED\_TYPE\_CODES = (

('numlock', 0x00),

('capslock', 0x01),

('scrolllock', 0x02),

('compose', 0x03),

('kana', 0x04),

('sleep', 0x05),

('suspend', 0x06),

('mute', 0x07),

('misc', 0x08),

('mail', 0x09),

('charging', 0x0a),

('max', 0x0f),

('cnt', 0x0f+1)

)

AUTOREPEAT\_VALUES = (

(0x00, "REP\_DELAY"),

(0x01, "REP\_PERIOD"),

(0x01, "REP\_MAX"),

(0x01+1, "REP\_CNT"))

SOUNDS = (

(0x00, "SND\_CLICK"),

(0x01, "SND\_BELL"),

(0x02, "SND\_TONE"),

(0x07, "SND\_MAX"),

(0x07+1, "SND\_CNT"))

WIN\_KEYBOARD\_CODES = {

0x0100: 1,

0x0101: 0,

0x104: 1,

0x105: 0,

}

WIN\_MOUSE\_CODES = {

0x0201: (0x110, 1, 589825), # WM\_LBUTTONDOWN --> BTN\_LEFT

0x0202: (0x110, 0, 589825), # WM\_LBUTTONUP --> BTN\_LEFT

0x0204: (0x111, 1, 589826), # WM\_RBUTTONDOWN --> BTN\_RIGHT

0x0205: (0x111, 0, 589826), # WM\_RBUTTONUP --> BTN\_RIGHT

0x0207: (0x112, 1, 589827), # WM\_MBUTTONDOWN --> BTN\_MIDDLE

0x0208: (0x112, 0, 589827), # WM\_MBUTTONU --> BTN\_MIDDLE

0x020B: (0x113, 1, 589828), # WM\_XBUTTONDOWN --> BTN\_SIDE

0x020C: (0x113, 0, 589828), # WM\_XBUTTONUP --> BTN\_SIDE

0x020B2: (0x114, 1, 589829), # WM\_XBUTTONDOWN --> BTN\_EXTRA

0x020C2: (0x114, 0, 589829), # WM\_XBUTTONUP --> BTN\_EXTRA

}

# THING SING That thing can sing!

# SONG LONG A long, long song.

# Good-bye, Thing. You sing too long.

# pylint: disable=too-many-lines

WINCODES = (

(0x01, 0x110), # Left mouse button

(0x02, 0x111), # Right mouse button

(0x03, 0), # Control-break processing

(0x04, 0x112), # Middle mouse button (three-button mouse)

(0x05, 0x113), # X1 mouse button

(0x06, 0x114), # X2 mouse button

(0x07, 0), # Undefined

(0x08, 14), # BACKSPACE key

(0x09, 15), # TAB key

(0x0A, 0), # Reserved

(0x0B, 0), # Reserved

(0x0C, 0x163), # CLEAR key

(0x0D, 28), # ENTER key

(0x0E, 0), # Undefined

(0x0F, 0), # Undefined

(0x10, 42), # SHIFT key

(0x11, 29), # CTRL key

(0x12, 56), # ALT key

(0x13, 119), # PAUSE key

(0x14, 58), # CAPS LOCK key

(0x15, 90), # IME Kana mode

(0x15, 91), # IME Hanguel mode (maintained for compatibility; use

# VK\_HANGUL)

(0x15, 91), # IME Hangul mode

(0x16, 0), # Undefined

(0x17, 92), # IME Junja mode - These all need to be fixed

(0x18, 93), # IME final mode - By someone who

(0x19, 94), # IME Hanja mode - Knows how

(0x19, 95), # IME Kanji mode - Japanese Keyboards work

(0x1A, 0), # Undefined

(0x1B, 1), # ESC key

(0x1C, 0), # IME convert

(0x1D, 0), # IME nonconvert

(0x1E, 0), # IME accept

(0x1F, 0), # IME mode change request

(0x20, 57), # SPACEBAR

(0x21, 104), # PAGE UP key

(0x22, 109), # PAGE DOWN key

(0x23, 107), # END key

(0x24, 102), # HOME key

(0x25, 105), # LEFT ARROW key

(0x26, 103), # UP ARROW key

(0x27, 106), # RIGHT ARROW key

(0x28, 108), # DOWN ARROW key

(0x29, 0x161), # SELECT key

(0x2A, 210), # PRINT key

(0x2B, 28), # EXECUTE key

(0x2C, 99), # PRINT SCREEN key

(0x2D, 110), # INS key

(0x2E, 111), # DEL key

(0x2F, 138), # HELP key

(0x30, 11), # 0 key

(0x31, 2), # 1 key

(0x32, 3), # 2 key

(0x33, 4), # 3 key

(0x34, 5), # 4 key

(0x35, 6), # 5 key

(0x36, 7), # 6 key

(0x37, 8), # 7 key

(0x38, 9), # 8 key

(0x39, 10), # 9 key

# (0x3A-40, 0), # Undefined

(0x41, 30), # A key

(0x42, 48), # B key

(0x43, 46), # C key

(0x44, 32), # D key

(0x45, 18), # E key

(0x46, 33), # F key

(0x47, 34), # G key

(0x48, 35), # H key

(0x49, 23), # I key

(0x4A, 36), # J key

(0x4B, 37), # K key

(0x4C, 38), # L key

(0x4D, 50), # M key

(0x4E, 49), # N key

(0x4F, 24), # O key

(0x50, 25), # P key

(0x51, 16), # Q key

(0x52, 19), # R key

(0x53, 31), # S key

(0x54, 20), # T key

(0x55, 22), # U key

(0x56, 47), # V key

(0x57, 17), # W key

(0x58, 45), # X key

(0x59, 21), # Y key

(0x5A, 44), # Z key

(0x5B, 125), # Left Windows key (Natural keyboard)

(0x5C, 126), # Right Windows key (Natural keyboard)

(0x5D, 139), # Applications key (Natural keyboard)

(0x5E, 0), # Reserved

(0x5F, 142), # Computer Sleep key

(0x60, 82), # Numeric keypad 0 key

(0x61, 79), # Numeric keypad 1 key

(0x62, 80), # Numeric keypad 2 key

(0x63, 81), # Numeric keypad 3 key

(0x64, 75), # Numeric keypad 4 key

(0x65, 76), # Numeric keypad 5 key

(0x66, 77), # Numeric keypad 6 key

(0x67, 71), # Numeric keypad 7 key

(0x68, 72), # Numeric keypad 8 key

(0x69, 73), # Numeric keypad 9 key

(0x6A, 55), # Multiply key

(0x6B, 78), # Add key

(0x6C, 96), # Separator key

(0x6D, 74), # Subtract key

(0x6E, 83), # Decimal key

(0x6F, 98), # Divide key

(0x70, 59), # F1 key

(0x71, 60), # F2 key

(0x72, 61), # F3 key

(0x73, 62), # F4 key

(0x74, 63), # F5 key

(0x75, 64), # F6 key

(0x76, 65), # F7 key

(0x77, 66), # F8 key

(0x78, 67), # F9 key

(0x79, 68), # F10 key

(0x7A, 87), # F11 key

(0x7B, 88), # F12 key

(0x7C, 183), # F13 key

(0x7D, 184), # F14 key

(0x7E, 185), # F15 key

(0x7F, 186), # F16 key

(0x80, 187), # F17 key

(0x81, 188), # F18 key

(0x82, 189), # F19 key

(0x83, 190), # F20 key

(0x84, 191), # F21 key

(0x85, 192), # F22 key

(0x86, 192), # F23 key

(0x87, 194), # F24 key

# (0x88-8F, 0), # Unassigned

(0x90, 69), # NUM LOCK key

(0x91, 70), # SCROLL LOCK key

# (0x92-96, 0), # OEM specific

# (0x97-9F, 0), # Unassigned

(0xA0, 42), # Left SHIFT key

(0xA1, 54), # Right SHIFT key

(0xA2, 29), # Left CONTROL key

(0xA3, 97), # Right CONTROL key

(0xA4, 125), # Left MENU key

(0xA5, 126), # Right MENU key

(0xA6, 158), # Browser Back key

(0xA7, 159), # Browser Forward key

(0xA8, 173), # Browser Refresh key

(0xA9, 128), # Browser Stop key

(0xAA, 217), # Browser Search key

(0xAB, 0x16c), # Browser Favorites key

(0xAC, 150), # Browser Start and Home key

(0xAD, 113), # Volume Mute key

(0xAE, 114), # Volume Down key

(0xAF, 115), # Volume Up key

(0xB0, 163), # Next Track key

(0xB1, 165), # Previous Track key

(0xB2, 166), # Stop Media key

(0xB3, 164), # Play/Pause Media key

(0xB4, 155), # Start Mail key

(0xB5, 0x161), # Select Media key

(0xB6, 148), # Start Application 1 key

(0xB7, 149), # Start Application 2 key

# (0xB8-B9, 0), # Reserved

(0xBA, 39), # Used for miscellaneous characters; it can vary by keyboard.

(0xBB, 13), # For any country/region, the '+' key

(0xBC, 51), # For any country/region, the ',' key

(0xBD, 12), # For any country/region, the '-' key

(0xBE, 52), # For any country/region, the '.' key

(0xBF, 53), # Slash

(0xC0, 40), # Apostrophe

# (0xC1-D7, 0), # Reserved

# (0xD8-DA, 0), # Unassigned

(0xDB, 26), # [

(0xDC, 86), # \

(0xDD, 27), # ]

(0xDE, 43), # '

(0xDF, 119), # VK\_OFF - What's that?

(0xE0, 0), # Reserved

(0xE1, 0), # OEM Specific

(0xE2, 43), # Either the angle bracket key or the backslash key

# on the RT 102-key keyboard (0xE3-E4, 0), # OEM

# specific

(0xE5, 0), # IME PROCESS key

(0xE6, 0), # OEM specific

(0xE7, 0), # Used to pass Unicode characters as if they were

# keystrokes. The VK\_PACKET key is the low word of a

# 32-bit Virtual Key value used for non-keyboard input

# methods. For more information, see Remark in

# KEYBDINPUT, SendInput, WM\_KEYDOWN, and WM\_KEYUP

(0xE8, 0), # Unassigned

# (0xE9-F5, 0), # OEM specific

(0xF6, 0), # Attn key

(0xF7, 0), # CrSel key

(0xF8, 0), # ExSel key

(0xF9, 222), # Erase EOF key

(0xFA, 207), # Play key

(0xFB, 0x174), # Zoom key

(0xFC, 0), # Reserved

(0xFD, 0x19b), # PA1 key

(0xFE, 0x163), # Clear key

(0xFF, 185)

)

MAC\_EVENT\_CODES = (

# NSLeftMouseDown Quartz.kCGEventLeftMouseDown

(1, ("Key", 0x110, 1, 589825)),

# NSLeftMouseUp Quartz.kCGEventLeftMouseUp

(2, ("Key", 0x110, 0, 589825)),

# NSRightMouseDown Quartz.kCGEventRightMouseDown

(3, ("Key", 0x111, 1, 589826)),

# NSRightMouseUp Quartz.kCGEventRightMouseUp

(4, ("Key", 0x111, 0, 589826)),

(5, (None, 0, 0, 0)), # NSMouseMoved Quartz.kCGEventMouseMoved

(6, (None, 0, 0, 0)), # NSLeftMouseDragged Quartz.kCGEventLeftMouseDragged

# NSRightMouseDragged Quartz.kCGEventRightMouseDragged

(7, (None, 0, 0, 0)),

(8, (None, 0, 0, 0)), # NSMouseEntered

(9, (None, 0, 0, 0)), # NSMouseExited

(10, (None, 0, 0, 0)), # NSKeyDown

(11, (None, 0, 0, 0)), # NSKeyUp

(12, (None, 0, 0, 0)), # NSFlagsChanged

(13, (None, 0, 0, 0)), # NSAppKitDefined

(14, (None, 0, 0, 0)), # NSSystemDefined

(15, (None, 0, 0, 0)), # NSApplicationDefined

(16, (None, 0, 0, 0)), # NSPeriodic

(17, (None, 0, 0, 0)), # NSCursorUpdate

(22, (None, 0, 0, 0)), # NSScrollWheel Quartz.kCGEventScrollWheel

(23, (None, 0, 0, 0)), # NSTabletPoint Quartz.kCGEventTabletPointer

(24, (None, 0, 0, 0)), # NSTabletProximity Quartz.kCGEventTabletProximity

(25, (None, 0, 0, 0)), # NSOtherMouseDown Quartz.kCGEventOtherMouseDown

(25.2, ("Key", 0x112, 1, 589827)), # BTN\_MIDDLE

(25.3, ("Key", 0x113, 1, 589828)), # BTN\_SIDE

(25.4, ("Key", 0x114, 1, 589829)), # BTN\_EXTRA

(26, (None, 0, 0, 0)), # NSOtherMouseUp Quartz.kCGEventOtherMouseUp

(26.2, ("Key", 0x112, 0, 589827)), # BTN\_MIDDLE

(26.3, ("Key", 0x113, 0, 589828)), # BTN\_SIDE

(26.4, ("Key", 0x114, 0, 589829)), # BTN\_EXTRA

(27, (None, 0, 0, 0)), # NSOtherMouseDragged

(29, (None, 0, 0, 0)), # NSEventTypeGesture

(30, (None, 0, 0, 0)), # NSEventTypeMagnify

(31, (None, 0, 0, 0)), # NSEventTypeSwipe

(18, (None, 0, 0, 0)), # NSEventTypeRotate

(19, (None, 0, 0, 0)), # NSEventTypeBeginGesture

(20, (None, 0, 0, 0)), # NSEventTypeEndGesture

(27, (None, 0, 0, 0)), # Quartz.kCGEventOtherMouseDragged

(32, (None, 0, 0, 0)), # NSEventTypeSmartMagnify

(33, (None, 0, 0, 0)), # NSEventTypeQuickLook

(34, (None, 0, 0, 0)), # NSEventTypePressure

)

MAC\_KEYS = (

(0x00, 30), # kVK\_ANSI\_A

(0x01, 31), # kVK\_ANSI\_S (0x02, 32), # kVK\_ANSI\_D

(0x03, 33), # kVK\_ANSI\_F

(0x04, 35), # kVK\_ANSI\_H

(0x05, 34), # kVK\_ANSI\_G

(0x06, 44), # kVK\_ANSI\_Z

(0x07, 45), # kVK\_ANSI\_X

(0x08, 46), # kVK\_ANSI\_C

(0x09, 47), # kVK\_ANSI\_V

(0x0B, 48), # kVK\_ANSI\_B

(0x0C, 16), # kVK\_ANSI\_Q

(0x0D, 17), # kVK\_ANSI\_W

(0x0E, 18), # kVK\_ANSI\_E

(0x0F, 33), # kVK\_ANSI\_R

(0x10, 21), # kVK\_ANSI\_Y

(0x11, 20), # kVK\_ANSI\_T

(0x12, 2), # kVK\_ANSI\_1

(0x13, 3), # kVK\_ANSI\_2

(0x14, 4), # kVK\_ANSI\_3

(0x15, 5), # kVK\_ANSI\_4

(0x16, 7), # kVK\_ANSI\_6

(0x17, 6), # kVK\_ANSI\_5

(0x18, 13), # kVK\_ANSI\_Equal

(0x19, 10), # kVK\_ANSI\_9

(0x1A, 8), # kVK\_ANSI\_7

(0x1B, 12), # kVK\_ANSI\_Minus

(0x1C, 9), # kVK\_ANSI\_8

(0x1D, 11), # kVK\_ANSI\_0

(0x1E, 27), # kVK\_ANSI\_RightBracket

(0x1F, 24), # kVK\_ANSI\_O

(0x20, 22), # kVK\_ANSI\_U

(0x21, 26), # kVK\_ANSI\_LeftBracket

(0x22, 23), # kVK\_ANSI\_I

(0x23, 25), # kVK\_ANSI\_P

(0x25, 38), # kVK\_ANSI\_L

(0x26, 36), # kVK\_ANSI\_J

(0x27, 40), # kVK\_ANSI\_Quote

(0x28, 37), # kVK\_ANSI\_K

(0x29, 39), # kVK\_ANSI\_Semicolon

(0x2A, 43), # kVK\_ANSI\_Backslash

(0x2B, 51), # kVK\_ANSI\_Comma

(0x2C, 53), # kVK\_ANSI\_Slash

(0x2D, 49), # kVK\_ANSI\_N

(0x2E, 50), # kVK\_ANSI\_M

(0x2F, 52), # kVK\_ANSI\_Period

(0x32, 41), # kVK\_ANSI\_Grave

(0x41, 83), # kVK\_ANSI\_KeypadDecimal

(0x43, 55), # kVK\_ANSI\_KeypadMultiply

(0x45, 78), # kVK\_ANSI\_KeypadPlus

(0x47, 69), # kVK\_ANSI\_KeypadClear

(0x4B, 98), # kVK\_ANSI\_KeypadDivide

(0x4C, 96), # kVK\_ANSI\_KeypadEnter

(0x4E, 74), # kVK\_ANSI\_KeypadMinus

(0x51, 117), # kVK\_ANSI\_KeypadEquals

(0x52, 82), # kVK\_ANSI\_Keypad0

(0x53, 79), # kVK\_ANSI\_Keypad1

(0x54, 80), # kVK\_ANSI\_Keypad2

(0x55, 81), # kVK\_ANSI\_Keypad3

(0x56, 75), # kVK\_ANSI\_Keypad4

(0x57, 76), # kVK\_ANSI\_Keypad5

(0x58, 77), # kVK\_ANSI\_Keypad6

(0x59, 71), # kVK\_ANSI\_Keypad7

(0x5B, 72), # kVK\_ANSI\_Keypad8

(0x5C, 73), # kVK\_ANSI\_Keypad9

(0x24, 28), # kVK\_Return

(0x30, 15), # kVK\_Tab

(0x31, 57), # kVK\_Space

(0x33, 111), # kVK\_Delete

(0x35, 1), # kVK\_Escape

(0x37, 125), # kVK\_Command

(0x38, 42), # kVK\_Shift

(0x39, 58), # kVK\_CapsLock

(0x3A, 56), # kVK\_Option

(0x3B, 29), # kVK\_Control

(0x3C, 54), # kVK\_RightShift

(0x3D, 100), # kVK\_RightOption

(0x3E, 126), # kVK\_RightControl

(0x36, 126), # Right Meta

(0x3F, 0x1d0), # kVK\_Function

(0x40, 187), # kVK\_F17

(0x48, 115), # kVK\_VolumeUp

(0x49, 114), # kVK\_VolumeDown

(0x4A, 113), # kVK\_Mute

(0x4F, 188), # kVK\_F18

(0x50, 189), # kVK\_F19

(0x5A, 190), # kVK\_F20

(0x60, 63), # kVK\_F5

(0x61, 64), # kVK\_F6

(0x62, 65), # kVK\_F7

(0x63, 61), # kVK\_F3

(0x64, 66), # kVK\_F8

(0x65, 67), # kVK\_F9

(0x67, 87), # kVK\_F11

(0x69, 183), # kVK\_F13

(0x6A, 186), # kVK\_F16

(0x6B, 184), # kVK\_F14

(0x6D, 68), # kVK\_F10

(0x6F, 88), # kVK\_F12

(0x71, 185), # kVK\_F15

(0x72, 138), # kVK\_Help

(0x73, 102), # kVK\_Home

(0x74, 104), # kVK\_PageUp

(0x75, 111), # kVK\_ForwardDelete

(0x76, 62), # kVK\_F4

(0x77, 107), # kVK\_End

(0x78, 60), # kVK\_F2

(0x79, 109), # kVK\_PageDown

(0x7A, 59), # kVK\_F1

(0x7B, 105), # kVK\_LeftArrow

(0x7C, 106), # kVK\_RightArrow

(0x7D, 108), # kVK\_DownArrow

(0x7E, 103), # kVK\_UpArrow

(0x0A, 170), # kVK\_ISO\_Section

(0x5D, 124), # kVK\_JIS\_Yen

(0x5E, 92), # kVK\_JIS\_Underscore

(0x5F, 95), # kVK\_JIS\_KeypadComma

(0x66, 94), # kVK\_JIS\_Eisu

(0x68, 90) # kVK\_JIS\_Kana

)

# We have yet to support force feedback but probably should

# eventually:

FORCE\_FEEDBACK = () # Motor in gamepad

FORCE\_FEEDBACK\_STATUS = () # Status of motor

POWER = () # Power switch

# These two are internal workings of evdev we probably will never care

# about.

MAX = ()

CURRENT = ()

EVENT\_MAP = (

('types', EVENT\_TYPES),

('type\_codes', ((value, key) for key, value in EVENT\_TYPES)),

('wincodes', WINCODES),

('specials', SPECIAL\_DEVICES),

('xpad', XINPUT\_MAPPING),

('Sync', SYNCHRONIZATION\_EVENTS),

('Key', KEYS\_AND\_BUTTONS),

('Relative', RELATIVE\_AXES),

('Absolute', ABSOLUTE\_AXES),

('Misc', MISC\_EVENTS),

('Switch', SWITCH\_EVENTS),

('LED', LEDS),

('LED\_type\_codes', LED\_TYPE\_CODES),

('Sound', SOUNDS),

('Repeat', AUTOREPEAT\_VALUES),

('ForceFeedback', FORCE\_FEEDBACK),

('Power', POWER),

('ForceFeedbackStatus', FORCE\_FEEDBACK\_STATUS),

('Max', MAX),

('Current', CURRENT))

# Evdev style paths for the Mac

APPKIT\_KB\_PATH = "/dev/input/by-id/usb-AppKit\_Keyboard-event-kbd"

QUARTZ\_MOUSE\_PATH = "/dev/input/by-id/usb-Quartz\_Mouse-event-mouse"

APPKIT\_MOUSE\_PATH = "/dev/input/by-id/usb-AppKit\_Mouse-event-mouse"

# Now comes all the structs we need to parse the infomation coming

# from Windows.

class KBDLLHookStruct(ctypes.Structure):

"""Contains information about a low-level keyboard input event.

For full details see Microsoft's documentation:

https://msdn.microsoft.com/en-us/library/windows/desktop/

ms644967%28v=vs.85%29.aspx

"""

# pylint: disable=too-few-public-methods

\_fields\_ = [("vk\_code", DWORD),

("scan\_code", DWORD),

("flags", DWORD),

("time", ctypes.c\_int)]

class MSLLHookStruct(ctypes.Structure):

"""Contains information about a low-level mouse input event.

For full details see Microsoft's documentation:

https://msdn.microsoft.com/en-us/library/windows/desktop/

ms644970%28v=vs.85%29.aspx

"""

# pylint: disable=too-few-public-methods

\_fields\_ = [("x\_pos", ctypes.c\_long),

("y\_pos", ctypes.c\_long),

('reserved', ctypes.c\_short),

('mousedata', ctypes.c\_short),

("flags", DWORD),

("time", DWORD),

("extrainfo", ctypes.c\_ulong)]

class NoDataError(RuntimeError):

"""We have no data available but the user wants to read non blocking"""

pass

class XinputGamepad(ctypes.Structure):

"""Describes the current state of the Xbox 360 Controller.

For full details see Microsoft's documentation:

https://msdn.microsoft.com/en-us/library/windows/desktop/

microsoft.directx\_sdk.reference.xinput\_gamepad%28v=vs.85%29.aspx

"""

# pylint: disable=too-few-public-methods

\_fields\_ = [

('buttons', ctypes.c\_ushort), # wButtons

('left\_trigger', ctypes.c\_ubyte), # bLeftTrigger

('right\_trigger', ctypes.c\_ubyte), # bLeftTrigger

('l\_thumb\_x', ctypes.c\_short), # sThumbLX

('l\_thumb\_y', ctypes.c\_short), # sThumbLY

('r\_thumb\_x', ctypes.c\_short), # sThumbRx

('r\_thumb\_y', ctypes.c\_short), # sThumbRy

]

class XinputState(ctypes.Structure):

"""Represents the state of a controller.

For full details see Microsoft's documentation:

https://msdn.microsoft.com/en-us/library/windows/desktop/

microsoft.directx\_sdk.reference.xinput\_state%28v=vs.85%29.aspx

"""

# pylint: disable=too-few-public-methods

\_fields\_ = [

('packet\_number', ctypes.c\_ulong), # dwPacketNumber

('gamepad', XinputGamepad), # Gamepad

]

class XinputVibration(ctypes.Structure):

"""Specifies motor speed levels for the vibration function of a

controller.

For full details see Microsoft's documentation:

https://msdn.microsoft.com/en-us/library/windows/desktop/

microsoft.directx\_sdk.reference.xinput\_vibration%28v=vs.85%29.aspx

"""

# pylint: disable=too-few-public-methods

\_fields\_ = [("wLeftMotorSpeed", ctypes.c\_ushort),

("wRightMotorSpeed", ctypes.c\_ushort)]

if sys.version\_info.major == 2:

# pylint: disable=redefined-builtin

class PermissionError(IOError):

"""Raised when trying to run an operation without the adequate access

rights - for example filesystem permissions. Corresponds to errno

EACCES and EPERM."""

class UnpluggedError(RuntimeError):

"""The device requested is not plugged in."""

pass

class NoDevicePath(RuntimeError):

"""No evdev device path was given."""

pass

class UnknownEventType(IndexError):

"""We don't know what this event is."""

pass

class UnknownEventCode(IndexError):

"""We don't know what this event is."""

pass

class InputEvent(object): # pylint: disable=useless-object-inheritance

"""A user event."""

# pylint: disable=too-few-public-methods

def \_\_init\_\_(self,

device,

event\_info):

self.device = device

self.timestamp = event\_info["timestamp"]

self.code = event\_info["code"]

self.state = event\_info["state"]

self.ev\_type = event\_info["ev\_type"]

class BaseListener(object): # pylint: disable=useless-object-inheritance

"""Loosely emulate Evdev keyboard behaviour on other platforms.

Listen (hook in Windows terminology) for key events then buffer

them in a pipe.

"""

def \_\_init\_\_(self, pipe, events=None, codes=None):

self.pipe = pipe

self.events = events if events else []

self.codes = codes if codes else None

self.app = None

self.timeval = None

self.type\_codes = dict((

(value, key)

for key, value in EVENT\_TYPES))

self.install\_handle\_input()

def install\_handle\_input(self):

"""Install the input handler."""

pass

def uninstall\_handle\_input(self):

"""Un-install the input handler."""

pass

def \_\_del\_\_(self):

"""Clean up when deleted."""

self.uninstall\_handle\_input()

@staticmethod

def get\_timeval():

"""Get the time in seconds and microseconds."""

return convert\_timeval(time.time())

def update\_timeval(self):

"""Update the timeval with the current time."""

self.timeval = self.get\_timeval()

def create\_event\_object(self,

event\_type,

code,

value,

timeval=None):

"""Create an evdev style structure."""

if not timeval:

self.update\_timeval()

timeval = self.timeval

try:

event\_code = self.type\_codes[event\_type]

except KeyError:

raise UnknownEventType(

"We don't know what kind of event a %s is." % event\_type)

event = struct.pack(EVENT\_FORMAT,

timeval[0],

timeval[1],

event\_code,

code,

value)

return event

def write\_to\_pipe(self, event\_list):

"""Send event back to the mouse object."""

self.pipe.send\_bytes(b''.join(event\_list))

def emulate\_wheel(self, data, direction, timeval):

"""Emulate rel values for the mouse wheel.

In evdev, a single click forwards of the mouse wheel is 1 and

a click back is -1. Windows uses 120 and -120. We floor divide

the Windows number by 120. This is fine for the digital scroll

wheels found on the vast majority of mice. It also works on

the analogue ball on the top of the Apple mouse.

What do the analogue scroll wheels found on 200 quid high end

gaming mice do? If the lowest unit is 120 then we are okay. If

they report changes of less than 120 units Windows, then this

might be an unacceptable loss of precision. Needless to say, I

don't have such a mouse to test one way or the other.

"""

if direction == 'x':

code = 0x06

elif direction == 'z':

# Not enitely sure if this exists

code = 0x07

else:

code = 0x08

if WIN:

data = data // 120

return self.create\_event\_object(

"Relative",

code,

data,

timeval)

def emulate\_rel(self, key\_code, value, timeval):

"""Emulate the relative changes of the mouse cursor."""

return self.create\_event\_object(

"Relative",

key\_code,

value,

timeval)

def emulate\_press(self, key\_code, scan\_code, value, timeval):

"""Emulate a button press.

Currently supports 5 buttons.

The Microsoft documentation does not define what happens with

a mouse with more than five buttons, and I don't have such a

mouse.

From reading the Linux sources, I guess evdev can support up

to 255 buttons.

Therefore, I guess we could support more buttons quite easily,

if we had any useful hardware.

"""

scan\_event = self.create\_event\_object(

"Misc",

0x04,

scan\_code,

timeval)

key\_event = self.create\_event\_object(

"Key",

key\_code,

value,

timeval)

return scan\_event, key\_event

def emulate\_repeat(self, value, timeval):

"""The repeat press of a key/mouse button, e.g. double click."""

repeat\_event = self.create\_event\_object(

"Repeat",

2,

value,

timeval)

return repeat\_event

def sync\_marker(self, timeval):

"""Separate groups of events."""

return self.create\_event\_object(

"Sync",

0,

0,

timeval)

def emulate\_abs(self, x\_val, y\_val, timeval):

"""Emulate the absolute co-ordinates of the mouse cursor."""

x\_event = self.create\_event\_object(

"Absolute",

0x00,

x\_val,

timeval)

y\_event = self.create\_event\_object(

"Absolute",

0x01,

y\_val,

timeval)

return x\_event, y\_event

class WindowsKeyboardListener(BaseListener):

"""Loosely emulate Evdev keyboard behaviour on Windows. Listen (hook

in Windows terminology) for key events then buffer them in a pipe.

"""

def \_\_init\_\_(self, pipe, codes=None):

self.pipe = pipe

self.hooked = None

self.pointer = None

super(WindowsKeyboardListener, self).\_\_init\_\_(pipe, codes)

@staticmethod

def listen():

"""Listen for keyboard input."""

msg = MSG()

ctypes.windll.user32.GetMessageA(ctypes.byref(msg), 0, 0, 0)

def get\_fptr(self):

"""Get the function pointer."""

cmpfunc = ctypes.CFUNCTYPE(ctypes.c\_int,

WPARAM,

LPARAM,

ctypes.POINTER(KBDLLHookStruct))

return cmpfunc(self.handle\_input)

def install\_handle\_input(self):

"""Install the hook."""

self.pointer = self.get\_fptr()

self.hooked = ctypes.windll.user32.SetWindowsHookExA(

13,

self.pointer,

ctypes.windll.kernel32.GetModuleHandleW(None),

0

)

if not self.hooked:

return False

return True

def uninstall\_handle\_input(self):

"""Remove the hook."""

if self.hooked is None:

return

ctypes.windll.user32.UnhookWindowsHookEx(self.hooked)

self.hooked = None

def handle\_input(self, ncode, wparam, lparam):

"""Process the key input."""

value = WIN\_KEYBOARD\_CODES[wparam]

scan\_code = lparam.contents.scan\_code

vk\_code = lparam.contents.vk\_code

self.update\_timeval()

events = []

# Add key event

scan\_key, key\_event = self.emulate\_press(

vk\_code, scan\_code, value, self.timeval)

events.append(scan\_key)

events.append(key\_event)

# End with a sync marker

events.append(self.sync\_marker(self.timeval))

# We are done

self.write\_to\_pipe(events)

return ctypes.windll.user32.CallNextHookEx(

self.hooked, ncode, wparam, lparam)

def keyboard\_process(pipe):

"""Single subprocess for reading keyboard events on Windows."""

keyboard = WindowsKeyboardListener(pipe)

keyboard.listen()

class WindowsMouseListener(BaseListener):

"""Loosely emulate Evdev mouse behaviour on Windows. Listen (hook

in Windows terminology) for key events then buffer them in a pipe.

"""

def \_\_init\_\_(self, pipe):

self.pipe = pipe

self.hooked = None

self.pointer = None

self.mouse\_codes = WIN\_MOUSE\_CODES

super(WindowsMouseListener, self).\_\_init\_\_(pipe)

@staticmethod

def listen():

"""Listen for mouse input."""

msg = MSG()

ctypes.windll.user32.GetMessageA(ctypes.byref(msg), 0, 0, 0)

def get\_fptr(self):

"""Get the function pointer."""

cmpfunc = ctypes.CFUNCTYPE(ctypes.c\_int,

WPARAM,

LPARAM,

ctypes.POINTER(MSLLHookStruct))

return cmpfunc(self.handle\_input)

def install\_handle\_input(self):

"""Install the hook."""

self.pointer = self.get\_fptr()

self.hooked = ctypes.windll.user32.SetWindowsHookExA(

14,

self.pointer,

ctypes.windll.kernel32.GetModuleHandleW(None),

0

)

if not self.hooked:

return False

return True

def uninstall\_handle\_input(self):

"""Remove the hook."""

if self.hooked is None:

return

ctypes.windll.user32.UnhookWindowsHookEx(self.hooked)

self.hooked = None

def handle\_input(self, ncode, wparam, lparam):

"""Process the key input."""

x\_pos = lparam.contents.x\_pos

y\_pos = lparam.contents.y\_pos

data = lparam.contents.mousedata

# This is how we can distinguish mouse 1 from mouse 2

# extrainfo = lparam.contents.extrainfo

# The way windows seems to do it is there is primary mouse

# and all other mouses report as mouse 2

# Also useful later will be to support the flags field

# flags = lparam.contents.flags

# This shows if the event was from a real device or whether it

# was injected somehow via software

self.emulate\_mouse(wparam, x\_pos, y\_pos, data)

# Give back control to Windows to wait for and process the

# next event

return ctypes.windll.user32.CallNextHookEx(

self.hooked, ncode, wparam, lparam)

def emulate\_mouse(self, key\_code, x\_val, y\_val, data):

"""Emulate the ev codes using the data Windows has given us.

Note that by default in Windows, to recognise a double click,

you just notice two clicks in a row within a reasonablely

short time period.

However, if the application developer sets the application

window's class style to CS\_DBLCLKS, the operating system will

notice the four button events (down, up, down, up), intercept

them and then send a single key code instead.

There are no such special double click codes on other

platforms, so not obvious what to do with them. It might be

best to just convert them back to four events.

Currently we do nothing.

((0x0203, 'WM\_LBUTTONDBLCLK'),

(0x0206, 'WM\_RBUTTONDBLCLK'),

(0x0209, 'WM\_MBUTTONDBLCLK'),

(0x020D, 'WM\_XBUTTONDBLCLK'))

"""

# Once again ignore Windows' relative time (since system

# startup) and use the absolute time (since epoch i.e. 1st Jan

# 1970).

self.update\_timeval()

events = []

if key\_code == 0x0200:

# We have a mouse move alone.

# So just pass through to below

pass

elif key\_code == 0x020A:

# We have a vertical mouse wheel turn

events.append(self.emulate\_wheel(data, 'y', self.timeval))

elif key\_code == 0x020E:

# We have a horizontal mouse wheel turn

# https://msdn.microsoft.com/en-us/library/windows/desktop/

# ms645614%28v=vs.85%29.aspx

events.append(self.emulate\_wheel(data, 'x', self.timeval))

else:

# We have a button press.

# Distinguish the second extra button

if key\_code == 0x020B and data == 2:

key\_code = 0x020B2

elif key\_code == 0x020C and data == 2:

key\_code = 0x020C2

# Get the mouse codes

code, value, scan\_code = self.mouse\_codes[key\_code]

# Add in the press events

scan\_event, key\_event = self.emulate\_press(

code, scan\_code, value, self.timeval)

events.append(scan\_event)

events.append(key\_event)

# Add in the absolute position of the mouse cursor

x\_event, y\_event = self.emulate\_abs(x\_val, y\_val, self.timeval)

events.append(x\_event)

events.append(y\_event)

# End with a sync marker

events.append(self.sync\_marker(self.timeval))

# We are done

self.write\_to\_pipe(events)

def mouse\_process(pipe):

"""Single subprocess for reading mouse events on Windows."""

mouse = WindowsMouseListener(pipe)

mouse.listen()

class QuartzMouseBaseListener(BaseListener):

"""Emulate evdev mouse behaviour on mac."""

def \_\_init\_\_(self, pipe):

super(QuartzMouseBaseListener, self).\_\_init\_\_(

pipe,

codes=dict(MAC\_EVENT\_CODES))

self.active = True

self.events = []

def \_get\_mouse\_button\_number(self, event):

"""Get the mouse button number from an event."""

raise NotImplementedError

def \_get\_click\_state(self, event):

"""The click state from an event."""

raise NotImplementedError

def \_get\_scroll(self, event):

"""The scroll values from an event."""

raise NotImplementedError

def \_get\_absolute(self, event):

"""Get abolute cursor location."""

raise NotImplementedError

def \_get\_relative(self, event):

"""Get the relative mouse movement."""

raise NotImplementedError

def handle\_button(self, event, event\_type):

"""Convert the button information from quartz into evdev format."""

# 0 for left

# 1 for right

# 2 for middle/center

# 3 for side

mouse\_button\_number = self.\_get\_mouse\_button\_number(event)

# Identify buttons 3,4,5

if event\_type in (25, 26):

event\_type = event\_type + (mouse\_button\_number \* 0.1)

# Add buttons to events

event\_type\_string, event\_code, value, scan = self.codes[event\_type]

if event\_type\_string == "Key":

scan\_event, key\_event = self.emulate\_press(

event\_code, scan, value, self.timeval)

self.events.append(scan\_event)

self.events.append(key\_event)

# doubleclick/n-click of button

click\_state = self.\_get\_click\_state(event)

repeat = self.emulate\_repeat(click\_state, self.timeval)

self.events.append(repeat)

def handle\_scrollwheel(self, event):

"""Handle the scrollwheel (it is a ball on the mighty mouse)."""

# relative Scrollwheel

scroll\_x, scroll\_y = self.\_get\_scroll(event)

if scroll\_x:

self.events.append(

self.emulate\_wheel(scroll\_x, 'x', self.timeval))

if scroll\_y:

self.events.append(

self.emulate\_wheel(scroll\_y, 'y', self.timeval))

def handle\_absolute(self, event):

"""Absolute mouse position on the screen."""

(x\_val, y\_val) = self.\_get\_absolute(event)

x\_event, y\_event = self.emulate\_abs(

int(x\_val),

int(y\_val),

self.timeval)

self.events.append(x\_event)

self.events.append(y\_event)

def handle\_relative(self, event):

"""Relative mouse movement."""

delta\_x, delta\_y = self.\_get\_relative(event)

if delta\_x:

self.events.append(

self.emulate\_rel(0x00,

delta\_x,

self.timeval))

if delta\_y:

self.events.append(

self.emulate\_rel(0x01,

delta\_y,

self.timeval))

# pylint: disable=unused-argument

def handle\_input(self, proxy, event\_type, event, refcon):

"""Handle an input event."""

self.update\_timeval()

self.events = []

if event\_type in (1, 2, 3, 4, 25, 26, 27):

self.handle\_button(event, event\_type)

if event\_type == 22:

self.handle\_scrollwheel(event)

# Add in the absolute position of the mouse cursor

self.handle\_absolute(event)

# Add in the relative position of the mouse cursor

self.handle\_relative(event)

# End with a sync marker

self.events.append(self.sync\_marker(self.timeval))

# We are done

self.write\_to\_pipe(self.events)

def quartz\_mouse\_process(pipe):

"""Single subprocess for reading mouse events on Mac using newer Quartz."""

# Quartz only on the mac, so don't warn about Quartz

# pylint: disable=import-error

import Quartz

# pylint: disable=no-member

class QuartzMouseListener(QuartzMouseBaseListener):

"""Loosely emulate Evdev mouse behaviour on the Macs.

Listen for key events then buffer them in a pipe.

"""

def install\_handle\_input(self):

"""Constants below listed at:

https://developer.apple.com/documentation/coregraphics/

cgeventtype?language=objc#topics

"""

# Keep Mac Names to make it easy to find the documentation

# pylint: disable=invalid-name

NSMachPort = Quartz.CGEventTapCreate(

Quartz.kCGSessionEventTap,

Quartz.kCGHeadInsertEventTap,

Quartz.kCGEventTapOptionDefault,

Quartz.CGEventMaskBit(Quartz.kCGEventLeftMouseDown) |

Quartz.CGEventMaskBit(Quartz.kCGEventLeftMouseUp) |

Quartz.CGEventMaskBit(Quartz.kCGEventRightMouseDown) |

Quartz.CGEventMaskBit(Quartz.kCGEventRightMouseUp) |

Quartz.CGEventMaskBit(Quartz.kCGEventMouseMoved) |

Quartz.CGEventMaskBit(Quartz.kCGEventLeftMouseDragged) |

Quartz.CGEventMaskBit(Quartz.kCGEventRightMouseDragged) |

Quartz.CGEventMaskBit(Quartz.kCGEventScrollWheel) |

Quartz.CGEventMaskBit(Quartz.kCGEventTabletPointer) |

Quartz.CGEventMaskBit(Quartz.kCGEventTabletProximity) |

Quartz.CGEventMaskBit(Quartz.kCGEventOtherMouseDown) |

Quartz.CGEventMaskBit(Quartz.kCGEventOtherMouseUp) |

Quartz.CGEventMaskBit(Quartz.kCGEventOtherMouseDragged),

self.handle\_input,

None)

CFRunLoopSourceRef = Quartz.CFMachPortCreateRunLoopSource(

None,

NSMachPort,

0)

CFRunLoopRef = Quartz.CFRunLoopGetCurrent()

Quartz.CFRunLoopAddSource(

CFRunLoopRef,

CFRunLoopSourceRef,

Quartz.kCFRunLoopDefaultMode)

Quartz.CGEventTapEnable(

NSMachPort,

True)

def listen(self):

"""Listen for quartz events."""

while self.active:

Quartz.CFRunLoopRunInMode(

Quartz.kCFRunLoopDefaultMode, 5, False)

def uninstall\_handle\_input(self):

self.active = False

def \_get\_mouse\_button\_number(self, event):

"""Get the mouse button number from an event."""

return Quartz.CGEventGetIntegerValueField(

event, Quartz.kCGMouseEventButtonNumber)

def \_get\_click\_state(self, event):

"""The click state from an event."""

return Quartz.CGEventGetIntegerValueField(

event, Quartz.kCGMouseEventClickState)

def \_get\_scroll(self, event):

"""The scroll values from an event."""

scroll\_y = Quartz.CGEventGetIntegerValueField(

event, Quartz.kCGScrollWheelEventDeltaAxis1)

scroll\_x = Quartz.CGEventGetIntegerValueField(

event, Quartz.kCGScrollWheelEventDeltaAxis2)

return scroll\_x, scroll\_y

def \_get\_absolute(self, event):

"""Get abolute cursor location."""

return Quartz.CGEventGetLocation(event)

def \_get\_relative(self, event):

"""Get the relative mouse movement."""

delta\_x = Quartz.CGEventGetIntegerValueField(

event, Quartz.kCGMouseEventDeltaX)

delta\_y = Quartz.CGEventGetIntegerValueField(

event, Quartz.kCGMouseEventDeltaY)

return delta\_x, delta\_y

mouse = QuartzMouseListener(pipe)

mouse.listen()

class AppKitMouseBaseListener(BaseListener):

"""Emulate evdev behaviour on the the Mac."""

def \_\_init\_\_(self, pipe, events=None):

super(AppKitMouseBaseListener, self).\_\_init\_\_(

pipe, events, codes=dict(MAC\_EVENT\_CODES))

@staticmethod

def \_get\_mouse\_button\_number(event):

"""Get the button number."""

return event.buttonNumber()

@staticmethod

def \_get\_absolute(event):

"""Get the absolute (pixel) location of the mouse cursor."""

return event.locationInWindow()

@staticmethod

def \_get\_event\_type(event):

"""Get the appkit event type of the event."""

return event.type()

@staticmethod

def \_get\_deltas(event):

"""Get the changes from the appkit event."""

delta\_x = round(event.deltaX())

delta\_y = round(event.deltaY())

delta\_z = round(event.deltaZ())

return delta\_x, delta\_y, delta\_z

def handle\_button(self, event, event\_type):

"""Handle mouse click."""

mouse\_button\_number = self.\_get\_mouse\_button\_number(event)

# Identify buttons 3,4,5

if event\_type in (25, 26):

event\_type = event\_type + (mouse\_button\_number \* 0.1)

# Add buttons to events

event\_type\_name, event\_code, value, scan = self.codes[event\_type]

if event\_type\_name == "Key":

scan\_event, key\_event = self.emulate\_press(

event\_code, scan, value, self.timeval)

self.events.append(scan\_event)

self.events.append(key\_event)

def handle\_absolute(self, event):

"""Absolute mouse position on the screen."""

point = self.\_get\_absolute(event)

x\_pos = round(point.x)

y\_pos = round(point.y)

x\_event, y\_event = self.emulate\_abs(x\_pos, y\_pos, self.timeval)

self.events.append(x\_event)

self.events.append(y\_event)

def handle\_scrollwheel(self, event):

"""Make endev from appkit scroll wheel event."""

delta\_x, delta\_y, delta\_z = self.\_get\_deltas(event)

if delta\_x:

self.events.append(

self.emulate\_wheel(delta\_x, 'x', self.timeval))

if delta\_y:

self.events.append(

self.emulate\_wheel(delta\_y, 'y', self.timeval))

if delta\_z:

self.events.append(

self.emulate\_wheel(delta\_z, 'z', self.timeval))

def handle\_relative(self, event):

"""Get the position of the mouse on the screen."""

delta\_x, delta\_y, delta\_z = self.\_get\_deltas(event)

if delta\_x:

self.events.append(

self.emulate\_rel(0x00,

delta\_x,

self.timeval))

if delta\_y:

self.events.append(

self.emulate\_rel(0x01,

delta\_y,

self.timeval))

if delta\_z:

self.events.append(

self.emulate\_rel(0x02,

delta\_z,

self.timeval))

def handle\_input(self, event):

"""Process the mouse event."""

self.update\_timeval()

self.events = []

code = self.\_get\_event\_type(event)

# Deal with buttons

self.handle\_button(event, code)

# Mouse wheel

if code == 22:

self.handle\_scrollwheel(event)

# Other relative mouse movements

else:

self.handle\_relative(event)

# Add in the absolute position of the mouse cursor

self.handle\_absolute(event)

# End with a sync marker

self.events.append(self.sync\_marker(self.timeval))

# We are done

self.write\_to\_pipe(self.events)

def appkit\_mouse\_process(pipe):

"""Single subprocess for reading mouse events on Mac using older AppKit."""

# pylint: disable=import-error,too-many-locals

# Note Objective C does not support a Unix style fork.

# So these imports have to be inside the child subprocess since

# otherwise the child process cannot use them.

# pylint: disable=no-member, no-name-in-module

from Foundation import NSObject

from AppKit import NSApplication, NSApp

from Cocoa import (NSEvent, NSLeftMouseDownMask,

NSLeftMouseUpMask, NSRightMouseDownMask,

NSRightMouseUpMask, NSMouseMovedMask,

NSLeftMouseDraggedMask,

NSRightMouseDraggedMask, NSMouseEnteredMask,

NSMouseExitedMask, NSScrollWheelMask,

NSOtherMouseDownMask, NSOtherMouseUpMask)

from PyObjCTools import AppHelper

import objc

class MacMouseSetup(NSObject):

"""Setup the handler."""

@objc.python\_method

def init\_with\_handler(self, handler):

"""

Init method that receives the write end of the pipe.

"""

# ALWAYS call the super's designated initializer.

# Also, make sure to re-bind "self" just in case it

# returns something else!

# pylint: disable=self-cls-assignment

self = super(MacMouseSetup, self).init()

self.handler = handler

# Unlike Python's \_\_init\_\_, initializers MUST return self,

# because they are allowed to return any object!

return self

# pylint: disable=invalid-name, unused-argument

def applicationDidFinishLaunching\_(self, notification):

"""Bind the listen method as the handler for mouse events."""

mask = (NSLeftMouseDownMask | NSLeftMouseUpMask |

NSRightMouseDownMask | NSRightMouseUpMask |

NSMouseMovedMask | NSLeftMouseDraggedMask |

NSRightMouseDraggedMask | NSScrollWheelMask |

NSMouseEnteredMask | NSMouseExitedMask |

NSOtherMouseDownMask | NSOtherMouseUpMask)

NSEvent.addGlobalMonitorForEventsMatchingMask\_handler\_(

mask, self.handler)

class MacMouseListener(AppKitMouseBaseListener):

"""Loosely emulate Evdev mouse behaviour on the Macs.

Listen for key events then buffer them in a pipe.

"""

def install\_handle\_input(self):

"""Install the hook."""

self.app = NSApplication.sharedApplication()

# pylint: disable=no-member

delegate = MacMouseSetup.alloc().init\_with\_handler(

self.handle\_input)

NSApp().setDelegate\_(delegate)

AppHelper.runEventLoop()

def \_\_del\_\_(self):

"""Stop the listener on deletion."""

AppHelper.stopEventLoop()

# pylint: disable=unused-variable

mouse = MacMouseListener(pipe, events=[])

class AppKitKeyboardListener(BaseListener):

"""Emulate an evdev keyboard on the Mac."""

def \_\_init\_\_(self, pipe):

super(AppKitKeyboardListener, self).\_\_init\_\_(

pipe, codes=dict(MAC\_KEYS))

@staticmethod

def \_get\_event\_key\_code(event):

"""Get the key code."""

return event.keyCode()

@staticmethod

def \_get\_event\_type(event):

"""Get the event type."""

return event.type()

@staticmethod

def \_get\_flag\_value(event):

"""Note, this may be able to be made more accurate,

i.e. handle two modifier keys at once."""

flags = event.modifierFlags()

if flags == 0x100:

value = 0

else:

value = 1

return value

def \_get\_key\_value(self, event, event\_type):

"""Get the key value."""

if event\_type == 10:

value = 1

elif event\_type == 11:

value = 0

elif event\_type == 12:

value = self.\_get\_flag\_value(event)

else:

value = -1

return value

def handle\_input(self, event):

"""Process they keyboard input."""

self.update\_timeval()

self.events = []

code = self.\_get\_event\_key\_code(event)

if code in self.codes:

new\_code = self.codes[code]

else:

new\_code = 0

event\_type = self.\_get\_event\_type(event)

value = self.\_get\_key\_value(event, event\_type)

scan\_event, key\_event = self.emulate\_press(

new\_code, code, value, self.timeval)

self.events.append(scan\_event)

self.events.append(key\_event)

# End with a sync marker

self.events.append(self.sync\_marker(self.timeval))

# We are done

self.write\_to\_pipe(self.events)

def mac\_keyboard\_process(pipe):

"""Single subprocesses for reading keyboard on Mac."""

# pylint: disable=import-error,too-many-locals

# Note Objective C does not support a Unix style fork.

# So these imports have to be inside the child subprocess since

# otherwise the child process cannot use them.

# pylint: disable=no-member, no-name-in-module

from AppKit import NSApplication, NSApp

from Foundation import NSObject

from Cocoa import (NSEvent, NSKeyDownMask, NSKeyUpMask,

NSFlagsChangedMask)

from PyObjCTools import AppHelper

import objc

class MacKeyboardSetup(NSObject):

"""Setup the handler."""

@objc.python\_method

def init\_with\_handler(self, handler):

"""

Init method that receives the write end of the pipe.

"""

# ALWAYS call the super's designated initializer.

# Also, make sure to re-bind "self" just in case it

# returns something else!

# pylint: disable=self-cls-assignment

self = super(MacKeyboardSetup, self).init()

self.handler = handler

# Unlike Python's \_\_init\_\_, initializers MUST return self,

# because they are allowed to return any object!

return self

# pylint: disable=invalid-name, unused-argument

def applicationDidFinishLaunching\_(self, notification):

"""Bind the handler to listen to keyboard events."""

mask = NSKeyDownMask | NSKeyUpMask | NSFlagsChangedMask

NSEvent.addGlobalMonitorForEventsMatchingMask\_handler\_(

mask, self.handler)

class MacKeyboardListener(AppKitKeyboardListener):

"""Loosely emulate Evdev keyboard behaviour on the Mac.

Listen for key events then buffer them in a pipe.

"""

def install\_handle\_input(self):

"""Install the hook."""

self.app = NSApplication.sharedApplication()

# pylint: disable=no-member

delegate = MacKeyboardSetup.alloc().init\_with\_handler(

self.handle\_input)

NSApp().setDelegate\_(delegate)

AppHelper.runEventLoop()

def \_\_del\_\_(self):

"""Stop the listener on deletion."""

AppHelper.stopEventLoop()

# pylint: disable=unused-variable

keyboard = MacKeyboardListener(pipe)

class InputDevice(object): # pylint: disable=useless-object-inheritance

"""A user input device."""

# pylint: disable=too-many-instance-attributes

def \_\_init\_\_(self, manager,

device\_path=None,

char\_path\_override=None,

read\_size=1):

self.blocking = True

self.read\_size = read\_size

self.manager = manager

self.\_\_pipe = None

self.\_listener = None

self.leds = None

if device\_path:

self.\_device\_path = device\_path

else:

self.\_set\_device\_path()

# We should by now have a device\_path

try:

if not self.\_device\_path:

raise NoDevicePath

except AttributeError:

raise NoDevicePath

self.protocol, \_, self.device\_type = self.\_get\_path\_infomation()

if char\_path\_override:

self.\_character\_device\_path = char\_path\_override

else:

self.\_character\_device\_path = os.path.realpath(self.\_device\_path)

self.\_character\_file = None

self.\_evdev = False

self.\_set\_evdev\_state()

self.name = "Unknown Device"

self.\_set\_name()

def \_set\_device\_path(self):

"""Set the device path, overridden on the MAC and Windows."""

pass

def \_set\_evdev\_state(self):

"""Set whether the device is a real evdev device."""

if NIX:

self.\_evdev = True

def \_set\_name(self):

if NIX:

with open("/sys/class/input/%s/device/name" %

self.get\_char\_name()) as name\_file:

self.name = name\_file.read().strip()

self.leds = []

def \_get\_path\_infomation(self):

"""Get useful infomation from the device path."""

long\_identifier = self.\_device\_path.split('/')[4]

protocol, remainder = long\_identifier.split('-', 1)

identifier, \_, device\_type = remainder.rsplit('-', 2)

return (protocol, identifier, device\_type)

def get\_char\_name(self):

"""Get short version of char device name."""

return self.\_character\_device\_path.split('/')[-1]

def get\_char\_device\_path(self):

"""Get the char device path."""

return self.\_character\_device\_path

def \_\_str\_\_(self):

try:

return self.name

except AttributeError:

return "Unknown Device"

def \_\_repr\_\_(self):

return '%s.%s("%s")' % (

self.\_\_module\_\_,

self.\_\_class\_\_.\_\_name\_\_,

self.\_device\_path)

@property

def \_character\_device(self):

if not self.\_character\_file:

if WIN:

self.\_character\_file = io.BytesIO()

return self.\_character\_file

try:

self.\_character\_file = io.open(

self.\_character\_device\_path, 'rb')

except PermissionError:

# Python 3

raise PermissionError(PERMISSIONS\_ERROR\_TEXT)

except IOError as err:

# Python 2

if err.errno == 13:

raise PermissionError(PERMISSIONS\_ERROR\_TEXT)

else:

raise

return self.\_character\_file

def \_\_iter\_\_(self):

while True:

event = self.\_do\_iter()

if event:

yield event

def \_get\_data(self, read\_size):

"""Get data from the character device."""

return self.\_character\_device.read(read\_size)

@staticmethod

def \_get\_target\_function():

"""Get the correct target function. This is only used by Windows

subclasses."""

return False

def \_get\_total\_read\_size(self):

"""How much event data to process at once."""

if self.read\_size:

read\_size = EVENT\_SIZE \* self.read\_size

else:

read\_size = EVENT\_SIZE

return read\_size

def \_do\_iter(self):

read\_size = self.\_get\_total\_read\_size()

data = self.\_get\_data(read\_size)

if not data:

return None

evdev\_objects = iter\_unpack(data)

events = [self.\_make\_event(\*event) for event in evdev\_objects]

return events

# pylint: disable=too-many-arguments

def \_make\_event(self, tv\_sec, tv\_usec, ev\_type, code, value):

"""Create a friendly Python object from an evdev style event."""

event\_type = self.manager.get\_event\_type(ev\_type)

eventinfo = {

"ev\_type": event\_type,

"state": value,

"timestamp": tv\_sec + (tv\_usec / 1000000),

"code": self.manager.get\_event\_string(event\_type, code)

}

return InputEvent(self, eventinfo)

def read(self, blocking=True):

self.blocking = blocking

"""Read the next input event."""

return next(iter(self))

@property

def \_pipe(self):

"""On Windows we use a pipe to emulate a Linux style character

buffer."""

if self.\_evdev:

return None

if not self.\_\_pipe:

target\_function = self.\_get\_target\_function()

if not target\_function:

return None

self.\_\_pipe, child\_conn = Pipe(duplex=False)

self.\_listener = Process(target=target\_function,

args=(child\_conn,), daemon=True)

self.\_listener.start()

return self.\_\_pipe

def \_\_del\_\_(self):

if 'WIN' in globals() or 'MAC' in globals():

if WIN or MAC:

if self.\_\_pipe:

self.\_listener.terminate()

class Keyboard(InputDevice):

"""A keyboard or other key-like device.

Original umapped scan code, followed by the important key info

followed by a sync.

"""

def \_set\_device\_path(self):

super(Keyboard, self).\_set\_device\_path()

if MAC:

self.\_device\_path = APPKIT\_KB\_PATH

def \_set\_name(self):

super(Keyboard, self).\_set\_name()

if WIN:

self.name = "Microsoft Keyboard"

elif MAC:

self.name = "AppKit Keyboard"

@staticmethod

def \_get\_target\_function():

"""Get the correct target function."""

if WIN:

return keyboard\_process

if MAC:

return mac\_keyboard\_process

return None

def \_get\_data(self, read\_size):

"""Get data from the character device."""

if NIX:

return super(Keyboard, self).\_get\_data(read\_size)

return self.\_pipe.recv\_bytes()

class Mouse(InputDevice):

"""A mouse or other pointing-like device.

"""

def \_set\_device\_path(self):

super(Mouse, self).\_set\_device\_path()

if MAC:

self.\_device\_path = APPKIT\_MOUSE\_PATH

def \_set\_name(self):

super(Mouse, self).\_set\_name()

if WIN:

self.name = "Microsoft Mouse"

elif MAC:

self.name = "AppKit Mouse"

@staticmethod

def \_get\_target\_function():

"""Get the correct target function."""

if WIN:

return mouse\_process

if MAC:

return appkit\_mouse\_process

return None

def \_get\_data(self, read\_size):

"""Get data from the character device."""

if NIX:

return super(Mouse, self).\_get\_data(read\_size)

return self.\_pipe.recv\_bytes()

class MightyMouse(Mouse):

"""A mouse or other pointing device on the Mac."""

def \_set\_device\_path(self):

super(MightyMouse, self).\_set\_device\_path()

if MAC:

self.\_device\_path = QUARTZ\_MOUSE\_PATH

def \_set\_name(self):

self.name = "Quartz Mouse"

@staticmethod

def \_get\_target\_function():

"""Get the correct target function."""

return quartz\_mouse\_process

def delay\_and\_stop(duration, dll, device\_number):

"""Stop vibration aka force feedback aka rumble on

Windows after duration miliseconds."""

xinput = getattr(ctypes.windll, dll)

time.sleep(duration/1000)

xinput\_set\_state = xinput.XInputSetState

xinput\_set\_state.argtypes = [

ctypes.c\_uint, ctypes.POINTER(XinputVibration)]

xinput\_set\_state.restype = ctypes.c\_uint

vibration = XinputVibration(0, 0)

xinput\_set\_state(device\_number, ctypes.byref(vibration))

# I made this GamePad class before Mouse and Keyboard above, and have

# learned a lot about Windows in the process. This can probably be

# simplified massively and made to match Mouse and Keyboard more.

class GamePad(InputDevice):

"""A gamepad or other joystick-like device."""

def \_\_init\_\_(self, manager, device\_path,

char\_path\_override=None):

super(GamePad, self).\_\_init\_\_(manager,

device\_path,

char\_path\_override)

self.\_write\_file = None

self.\_\_device\_number = None

if WIN:

if "Microsoft\_Corporation\_Controller" in self.\_device\_path:

self.name = "Microsoft X-Box 360 pad"

identifier = self.\_get\_path\_infomation()[1]

self.\_\_device\_number = int(identifier.split('\_')[-1])

self.\_\_received\_packets = 0

self.\_\_missed\_packets = 0

self.\_\_last\_state = self.\_\_read\_device()

if NIX:

self.\_number\_xpad()

def \_number\_xpad(self):

"""Get the number of the joystick."""

js\_path = self.\_device\_path.replace('-event', '')

js\_chardev = os.path.realpath(js\_path)

try:

number\_text = js\_chardev.split('js')[1]

except IndexError:

return

try:

number = int(number\_text)

except ValueError:

return

self.\_\_device\_number = number

def get\_number(self):

"""Return the joystick number of the gamepad."""

return self.\_\_device\_number

def \_\_iter\_\_(self):

while True:

if WIN:

self.\_\_check\_state()

event = self.\_do\_iter()

if event:

yield event

else:

if not self.blocking:

raise NoDataError

def \_\_check\_state(self):

"""On Windows, check the state and fill the event character device."""

state = self.\_\_read\_device()

if not state:

raise UnpluggedError(

"Gamepad %d is not connected" % self.\_\_device\_number)

if state.packet\_number != self.\_\_last\_state.packet\_number:

# state has changed, handle the change

self.\_\_handle\_changed\_state(state)

self.\_\_last\_state = state

@staticmethod

def \_\_get\_timeval():

"""Get the time and make it into C style timeval."""

return convert\_timeval(time.time())

def create\_event\_object(self,

event\_type,

code,

value,

timeval=None):

"""Create an evdev style object."""

if not timeval:

timeval = self.\_\_get\_timeval()

try:

event\_code = self.manager.codes['type\_codes'][event\_type]

except KeyError:

raise UnknownEventType(

"We don't know what kind of event a %s is." % event\_type)

event = struct.pack(EVENT\_FORMAT,

timeval[0],

timeval[1],

event\_code,

code,

value)

return event

def \_\_write\_to\_character\_device(self, event\_list, timeval=None):

"""Emulate the Linux character device on other platforms such as

Windows."""

# Remember the position of the stream

pos = self.\_character\_device.tell()

# Go to the end of the stream

self.\_character\_device.seek(0, 2)

# Write the new data to the end

for event in event\_list:

self.\_character\_device.write(event)

# Add a sync marker

sync = self.create\_event\_object("Sync", 0, 0, timeval)

self.\_character\_device.write(sync)

# Put the stream back to its original position

self.\_character\_device.seek(pos)

def \_\_handle\_changed\_state(self, state):

"""

we need to pack a struct with the following five numbers:

tv\_sec, tv\_usec, ev\_type, code, value

then write it using \_\_write\_to\_character\_device

seconds, mircroseconds, ev\_type, code, value

time we just use now

ev\_type we look up

code we look up

value is 0 or 1 for the buttons

axis value is maybe the same as Linux? Hope so!

"""

timeval = self.\_\_get\_timeval()

events = self.\_\_get\_button\_events(state, timeval)

events.extend(self.\_\_get\_axis\_events(state, timeval))

if events:

self.\_\_write\_to\_character\_device(events, timeval)

def \_\_map\_button(self, button):

"""Get the linux xpad code from the Windows xinput code."""

\_, start\_code, start\_value = button

value = start\_value

ev\_type = "Key"

code = self.manager.codes['xpad'][start\_code]

if 1 <= start\_code <= 4:

ev\_type = "Absolute"

if start\_code == 1 and start\_value == 1:

value = -1

elif start\_code == 3 and start\_value == 1:

value = -1

return code, value, ev\_type

def \_\_map\_axis(self, axis):

"""Get the linux xpad code from the Windows xinput code."""

start\_code, start\_value = axis

value = start\_value

code = self.manager.codes['xpad'][start\_code]

return code, value

def \_\_get\_button\_events(self, state, timeval=None):

"""Get the button events from xinput."""

changed\_buttons = self.\_\_detect\_button\_events(state)

events = self.\_\_emulate\_buttons(changed\_buttons, timeval)

return events

def \_\_get\_axis\_events(self, state, timeval=None):

"""Get the stick events from xinput."""

axis\_changes = self.\_\_detect\_axis\_events(state)

events = self.\_\_emulate\_axis(axis\_changes, timeval)

return events

def \_\_emulate\_axis(self, axis\_changes, timeval=None):

"""Make the axis events use the Linux style format."""

events = []

for axis in axis\_changes:

code, value = self.\_\_map\_axis(axis)

event = self.create\_event\_object(

"Absolute",

code,

value,

timeval=timeval)

events.append(event)

return events

def \_\_emulate\_buttons(self, changed\_buttons, timeval=None):

"""Make the button events use the Linux style format."""

events = []

for button in changed\_buttons:

code, value, ev\_type = self.\_\_map\_button(button)

event = self.create\_event\_object(

ev\_type,

code,

value,

timeval=timeval)

events.append(event)

return events

@staticmethod

def \_\_gen\_bit\_values(number):

"""

Return a zero or one for each bit of a numeric value up to the most

significant 1 bit, beginning with the least significant bit.

"""

number = int(number)

while number:

yield number & 0x1

number >>= 1

def \_\_get\_bit\_values(self, number, size=32):

"""Get bit values as a list for a given number

>>> get\_bit\_values(1) == [0]\*31 + [1]

True

>>> get\_bit\_values(0xDEADBEEF)

[1L, 1L, 0L, 1L, 1L, 1L, 1L,

0L, 1L, 0L, 1L, 0L, 1L, 1L, 0L, 1L, 1L, 0L, 1L, 1L, 1L, 1L,

1L, 0L, 1L, 1L, 1L, 0L, 1L, 1L, 1L, 1L]

You may override the default word size of 32-bits to match your actual

application.

>>> get\_bit\_values(0x3, 2)

[1L, 1L]

>>> get\_bit\_values(0x3, 4)

[0L, 0L, 1L, 1L]

"""

res = list(self.\_\_gen\_bit\_values(number))

res.reverse()

# 0-pad the most significant bit

res = [0] \* (size - len(res)) + res

return res

def \_\_detect\_button\_events(self, state):

changed = state.gamepad.buttons ^ self.\_\_last\_state.gamepad.buttons

changed = self.\_\_get\_bit\_values(changed, 16)

buttons\_state = self.\_\_get\_bit\_values(state.gamepad.buttons, 16)

changed.reverse()

buttons\_state.reverse()

button\_numbers = count(1)

changed\_buttons = list(

filter(itemgetter(0),

list(zip(changed, button\_numbers, buttons\_state))))

# returns for example [(1,15,1)] type, code, value?

return changed\_buttons

def \_\_detect\_axis\_events(self, state):

# axis fields are everything but the buttons

# pylint: disable=protected-access

# Attribute name \_fields\_ is special name set by ctypes

axis\_fields = dict(XinputGamepad.\_fields\_)

axis\_fields.pop('buttons')

changed\_axes = []

# Ax\_type might be useful when we support high-level deadzone

# methods.

# pylint: disable=unused-variable

for axis, ax\_type in list(axis\_fields.items()):

old\_val = getattr(self.\_\_last\_state.gamepad, axis)

new\_val = getattr(state.gamepad, axis)

if old\_val != new\_val:

changed\_axes.append((axis, new\_val))

return changed\_axes

def \_\_read\_device(self):

"""Read the state of the gamepad."""

state = XinputState()

res = self.manager.xinput.XInputGetState(

self.\_\_device\_number, ctypes.byref(state))

if res == XINPUT\_ERROR\_SUCCESS:

return state

if res != XINPUT\_ERROR\_DEVICE\_NOT\_CONNECTED:

raise RuntimeError(

"Unknown error %d attempting to get state of device %d" % (

res, self.\_\_device\_number))

# else (device is not connected)

return None

@property

def \_write\_device(self):

if not self.\_write\_file:

if not NIX:

return None

try:

self.\_write\_file = io.open(

self.\_character\_device\_path, 'wb')

except PermissionError:

# Python 3

raise PermissionError(PERMISSIONS\_ERROR\_TEXT)

except IOError as err:

# Python 2

if err.errno == 13:

raise PermissionError(PERMISSIONS\_ERROR\_TEXT)

else:

raise

return self.\_write\_file

def \_start\_vibration\_win(self, left\_motor, right\_motor):

"""Start the vibration, which will run until stopped."""

xinput\_set\_state = self.manager.xinput.XInputSetState

xinput\_set\_state.argtypes = [

ctypes.c\_uint, ctypes.POINTER(XinputVibration)]

xinput\_set\_state.restype = ctypes.c\_uint

vibration = XinputVibration(

int(left\_motor \* 65535), int(right\_motor \* 65535))

xinput\_set\_state(self.\_\_device\_number, ctypes.byref(vibration))

def \_stop\_vibration\_win(self):

"""Stop the vibration."""

xinput\_set\_state = self.manager.xinput.XInputSetState

xinput\_set\_state.argtypes = [

ctypes.c\_uint, ctypes.POINTER(XinputVibration)]

xinput\_set\_state.restype = ctypes.c\_uint

stop\_vibration = ctypes.byref(XinputVibration(0, 0))

xinput\_set\_state(self.\_\_device\_number, stop\_vibration)

def \_set\_vibration\_win(self, left\_motor, right\_motor, duration):

"""Control the motors on Windows."""

self.\_start\_vibration\_win(left\_motor, right\_motor)

stop\_process = Process(target=delay\_and\_stop,

args=(duration,

self.manager.xinput\_dll,

self.\_\_device\_number))

stop\_process.start()

def \_\_get\_vibration\_code(self, left\_motor, right\_motor, duration):

"""This is some crazy voodoo, if you can simplify it, please do."""

inner\_event = struct.pack(

'2h6x2h2x2H28x',

0x50,

-1,

duration,

0,

int(left\_motor \* 65535),

int(right\_motor \* 65535))

buf\_conts = ioctl(self.\_write\_device, 1076905344, inner\_event)

return int(codecs.encode(buf\_conts[1:3], 'hex'), 16)

def \_set\_vibration\_nix(self, left\_motor, right\_motor, duration):

"""Control the motors on Linux.

Duration is in miliseconds."""

code = self.\_\_get\_vibration\_code(left\_motor, right\_motor, duration)

secs, msecs = convert\_timeval(time.time())

outer\_event = struct.pack(EVENT\_FORMAT, secs, msecs, 0x15, code, 1)

self.\_write\_device.write(outer\_event)

self.\_write\_device.flush()

def set\_vibration(self, left\_motor, right\_motor, duration):

"""Control the speed of both motors seperately or together.

left\_motor and right\_motor arguments require a number between

0 (off) and 1 (full).

duration is miliseconds, e.g. 1000 for a second."""

if WIN:

self.\_set\_vibration\_win(left\_motor, right\_motor, duration)

elif NIX:

self.\_set\_vibration\_nix(left\_motor, right\_motor, duration)

else:

raise NotImplementedError

class OtherDevice(InputDevice):

"""A device of which its is type is either undetectable or has not

been implemented yet.

"""

pass

class LED(object): # pylint: disable=useless-object-inheritance

"""A light source."""

def \_\_init\_\_(self, manager, path, name):

self.manager = manager

self.path = path

self.name = name

self.\_write\_file = None

self.\_character\_device\_path = None

self.\_post\_init()

def \_post\_init(self):

"""Post init setup."""

pass

def \_\_str\_\_(self):

return self.name

def \_\_repr\_\_(self):

return '%s.%s("%s")' % (

self.\_\_module\_\_,

self.\_\_class\_\_.\_\_name\_\_,

self.path)

def status(self):

"""Get the device status, i.e. the brightness level."""

status\_filename = os.path.join(self.path, 'brightness')

with open(status\_filename) as status\_fp:

result = status\_fp.read()

status\_text = result.strip()

try:

status = int(status\_text)

except ValueError:

return status\_text

return status

def max\_brightness(self):

"""Get the device's maximum brightness level."""

status\_filename = os.path.join(self.path, 'max\_brightness')

with open(status\_filename) as status\_fp:

result = status\_fp.read()

status\_text = result.strip()

try:

status = int(status\_text)

except ValueError:

return status\_text

return status

@property

def \_write\_device(self):

"""The output device."""

if not self.\_write\_file:

if not NIX:

return None

try:

self.\_write\_file = io.open(

self.\_character\_device\_path, 'wb')

except PermissionError:

# Python 3

raise PermissionError(PERMISSIONS\_ERROR\_TEXT)

except IOError as err:

# Python 2 only

if err.errno == 13: # pragma: no cover

raise PermissionError(PERMISSIONS\_ERROR\_TEXT)

else:

raise

return self.\_write\_file

def \_make\_event(self, event\_type, code, value):

"""Make a new event and send it to the character device."""

secs, msecs = convert\_timeval(time.time())

data = struct.pack(EVENT\_FORMAT,

secs,

msecs,

event\_type,

code,

value)

self.\_write\_device.write(data)

self.\_write\_device.flush()

class SystemLED(LED):

"""An LED on your system e.g. caps lock."""

def \_\_init\_\_(self, manager, path, name):

self.code = None

self.device\_path = None

self.device = None

super(SystemLED, self).\_\_init\_\_(manager, path, name)

def \_post\_init(self):

"""Set up the device path and type code."""

self.\_led\_type\_code = self.manager.get\_typecode('LED')

self.device\_path = os.path.realpath(os.path.join(self.path, 'device'))

if '::' in self.name:

chardev, code\_name = self.name.split('::')

if code\_name in self.manager.codes['LED\_type\_codes']:

self.code = self.manager.codes['LED\_type\_codes'][code\_name]

try:

event\_number = chardev.split('input')[1]

except IndexError:

print("Failed with", self.name)

raise

else:

self.\_character\_device\_path = '/dev/input/event' + event\_number

self.\_match\_device()

def on(self): # pylint: disable=invalid-name

"""Turn the light on."""

self.\_make\_event(1)

def off(self):

"""Turn the light off."""

self.\_make\_event(0)

def \_make\_event(self, value): # pylint: disable=arguments-differ

"""Make a new event and send it to the character device."""

super(SystemLED, self).\_make\_event(

self.\_led\_type\_code,

self.code,

value)

def \_match\_device(self):

"""If the LED is connected to an input device,

associate the objects."""

for device in self.manager.all\_devices:

if (device.get\_char\_device\_path() ==

self.\_character\_device\_path):

self.device = device

device.leds.append(self)

break

class GamepadLED(LED):

"""A light source on a gamepad."""

def \_\_init\_\_(self, manager, path, name):

self.code = None

self.device = None

self.gamepad = None

super(GamepadLED, self).\_\_init\_\_(manager, path, name)

def \_post\_init(self):

self.\_match\_device()

self.\_character\_device\_path = self.gamepad.get\_char\_device\_path()

def \_match\_device(self):

number = int(self.name.split('xpad')[1])

for gamepad in self.manager.gamepads:

if number == gamepad.get\_number():

self.gamepad = gamepad

gamepad.leds.append(self)

break

class RawInputDeviceList(ctypes.Structure):

"""

Contains information about a raw input device.

For full details see Microsoft's documentation:

http://msdn.microsoft.com/en-us/library/windows/desktop/

ms645568(v=vs.85).aspx

"""

# pylint: disable=too-few-public-methods

\_fields\_ = [

("hDevice", HANDLE),

("dwType", DWORD)

]

class DeviceManager(object): # pylint: disable=useless-object-inheritance

"""Provides access to all connected and detectible user input

devices."""

# pylint: disable=too-many-instance-attributes

def \_\_init\_\_(self):

self.codes = {key: dict(value) for key, value in EVENT\_MAP}

self.\_raw = []

self.keyboards = []

self.mice = []

self.gamepads = []

self.other\_devices = []

self.all\_devices = []

self.leds = []

self.microbits = []

self.xinput = None

self.xinput\_dll = None

if WIN:

self.\_raw\_device\_counts = {

'mice': 0,

'keyboards': 0,

'otherhid': 0,

'unknown': 0

}

self.\_post\_init()

def \_post\_init(self):

"""Call the find devices method for the relevant platform."""

if WIN:

self.\_find\_devices\_win()

elif MAC:

self.\_find\_devices\_mac()

else:

self.\_find\_devices()

self.\_update\_all\_devices()

if NIX:

self.\_find\_leds()

def \_update\_all\_devices(self):

"""Update the all\_devices list."""

self.all\_devices = []

self.all\_devices.extend(self.keyboards)

self.all\_devices.extend(self.mice)

self.all\_devices.extend(self.gamepads)

self.all\_devices.extend(self.other\_devices)

def \_parse\_device\_path(self, device\_path, char\_path\_override=None):

"""Parse each device and add to the approriate list."""

# 1. Make sure that we can parse the device path.

try:

device\_type = device\_path.rsplit('-', 1)[1]

except IndexError:

warn("The following device path was skipped as it could "

"not be parsed: %s" % device\_path, RuntimeWarning)

return

# 2. Make sure each device is only added once.

realpath = os.path.realpath(device\_path)

if realpath in self.\_raw:

return

self.\_raw.append(realpath)

# 3. All seems good, append the device to the relevant list.

if device\_type == 'kbd':

self.keyboards.append(Keyboard(self, device\_path,

char\_path\_override))

elif device\_type == 'mouse':

self.mice.append(Mouse(self, device\_path,

char\_path\_override))

elif device\_type == 'joystick':

self.gamepads.append(GamePad(self,

device\_path,

char\_path\_override))

else:

self.other\_devices.append(OtherDevice(self,

device\_path,

char\_path\_override))

def \_find\_xinput(self):

"""Find most recent xinput library."""

for dll in XINPUT\_DLL\_NAMES:

try:

self.xinput = getattr(ctypes.windll, dll)

except OSError:

pass

else:

# We found an xinput driver

self.xinput\_dll = dll

break

else:

# We didn't find an xinput library

warn(

"No xinput driver dll found, gamepads not supported.",

RuntimeWarning)

def \_find\_devices\_win(self):

"""Find devices on Windows."""

self.\_find\_xinput()

self.\_detect\_gamepads()

self.\_count\_devices()

if self.\_raw\_device\_counts['keyboards'] > 0:

self.keyboards.append(Keyboard(

self,

"/dev/input/by-id/usb-A\_Nice\_Keyboard-event-kbd"))

if self.\_raw\_device\_counts['mice'] > 0:

self.mice.append(Mouse(

self,

"/dev/input/by-id/usb-A\_Nice\_Mouse\_called\_Arthur-event-mouse"))

def \_find\_devices\_mac(self):

"""Find devices on Mac."""

self.keyboards.append(Keyboard(self))

self.mice.append(MightyMouse(self))

self.mice.append(Mouse(self))

def \_detect\_gamepads(self):

"""Find gamepads."""

state = XinputState()

# Windows allows up to 4 gamepads.

for device\_number in range(4):

res = self.xinput.XInputGetState(

device\_number, ctypes.byref(state))

if res == XINPUT\_ERROR\_SUCCESS:

# We found a gamepad

device\_path = (

"/dev/input/by\_id/" +

"usb-Microsoft\_Corporation\_Controller\_%s-event-joystick"

% device\_number)

self.gamepads.append(GamePad(self, device\_path))

continue

if res != XINPUT\_ERROR\_DEVICE\_NOT\_CONNECTED:

raise RuntimeError(

"Unknown error %d attempting to get state of device %d"

% (res, device\_number))

def \_count\_devices(self):

"""See what Windows' GetRawInputDeviceList wants to tell us.

For now, we are just seeing if there is at least one keyboard

and/or mouse attached.

GetRawInputDeviceList could be used to help distinguish between

different keyboards and mice on the system in the way Linux

can. However, Roma uno die non est condita.

"""

number\_of\_devices = ctypes.c\_uint()

if ctypes.windll.user32.GetRawInputDeviceList(

ctypes.POINTER(ctypes.c\_int)(),

ctypes.byref(number\_of\_devices),

ctypes.sizeof(RawInputDeviceList)) == -1:

warn("Call to GetRawInputDeviceList was unsuccessful."

"We have no idea if a mouse or keyboard is attached.",

RuntimeWarning)

return

devices\_found = (RawInputDeviceList \* number\_of\_devices.value)()

if ctypes.windll.user32.GetRawInputDeviceList(

devices\_found,

ctypes.byref(number\_of\_devices),

ctypes.sizeof(RawInputDeviceList)) == -1:

warn("Call to GetRawInputDeviceList was unsuccessful."

"We have no idea if a mouse or keyboard is attached.",

RuntimeWarning)

return

for device in devices\_found:

if device.dwType == 0:

self.\_raw\_device\_counts['mice'] += 1

elif device.dwType == 1:

self.\_raw\_device\_counts['keyboards'] += 1

elif device.dwType == 2:

self.\_raw\_device\_counts['otherhid'] += 1

else:

self.\_raw\_device\_counts['unknown'] += 1

def \_find\_devices(self):

"""Find available devices."""

self.\_find\_by('id')

self.\_find\_by('path')

self.\_find\_special()

def \_find\_by(self, key):

"""Find devices."""

by\_path = glob.glob('/dev/input/by-{key}/\*-event-\*'.format(key=key))

for device\_path in by\_path:

self.\_parse\_device\_path(device\_path)

def \_find\_leds(self):

"""Find LED devices, Linux-only so far."""

for path in glob.glob('/sys/class/leds/\*'):

self.\_parse\_led\_path(path)

def \_parse\_led\_path(self, path):

name = path.rsplit('/', 1)[1]

if name.startswith('xpad'):

self.leds.append(GamepadLED(self, path, name))

elif name.startswith('input'):

self.leds.append(SystemLED(self, path, name))

else:

self.leds.append(LED(self, path, name))

def \_get\_char\_names(self):

"""Get a list of already found devices."""

return [device.get\_char\_name() for

device in self.all\_devices]

def \_find\_special(self):

"""Look for special devices."""

charnames = self.\_get\_char\_names()

for eventdir in glob.glob('/sys/class/input/event\*'):

char\_name = os.path.split(eventdir)[1]

if char\_name in charnames:

continue

name\_file = os.path.join(eventdir, 'device', 'name')

with open(name\_file) as name\_file:

device\_name = name\_file.read().strip()

if device\_name in self.codes['specials']:

self.\_parse\_device\_path(

self.codes['specials'][device\_name],

os.path.join('/dev/input', char\_name))

def \_\_iter\_\_(self):

return iter(self.all\_devices)

def \_\_getitem\_\_(self, index):

try:

return self.all\_devices[index]

except IndexError:

raise IndexError("list index out of range")

def get\_event\_type(self, raw\_type):

"""Convert the code to a useful string name."""

try:

return self.codes['types'][raw\_type]

except KeyError:

raise UnknownEventType("We don't know this event type")

def get\_event\_string(self, evtype, code):

"""Get the string name of the event."""

if WIN and evtype == 'Key':

# If we can map the code to a common one then do it

try:

code = self.codes['wincodes'][code]

except KeyError:

pass

try:

return self.codes[evtype][code]

except KeyError:

raise UnknownEventCode("We don't know this event.", evtype, code)

def get\_typecode(self, name):

"""Returns type code for `name`."""

return self.codes['type\_codes'][name]

def detect\_microbit(self):

"""Detect a microbit."""

try:

gpad = MicroBitPad(self)

except ModuleNotFoundError:

warn(

"The microbit library could not be found in the pythonpath. \n"

"For more information, please visit \n"

"https://inputs.readthedocs.io/en/latest/user/microbit.html",

RuntimeWarning)

else:

self.microbits.append(gpad)

self.gamepads.append(gpad)

SPIN\_UP\_MOTOR = (

'00000', '00001', '00011', '00111', '01111', '11111', '01111', '00011',

'00001', '00000', '00001', '00011', '00111', '01111', '11111', '00000',

'11111', '00000', '11111', '00000',

)

class MicroBitPad(GamePad):

"""A BBC Micro:bit flashed with bitio."""

def \_\_init\_\_(self, manager, device\_path=None,

char\_path\_override=None):

if not device\_path:

device\_path = '/dev/input/by-id/dialup-BBC\_MicroBit-event-joystick'

if not char\_path\_override:

char\_path\_override = '/dev/input/microbit0'

super(MicroBitPad, self).\_\_init\_\_(manager,

device\_path,

char\_path\_override)

# pylint: disable=no-member,import-error

import microbit

self.microbit = microbit

self.default\_image = microbit.Image("00500:00500:00500:00500:00500")

self.\_setup\_rumble()

self.set\_display()

def set\_display(self, index=None):

"""Show an image on the display."""

# pylint: disable=no-member

if index:

image = self.microbit.Image.STD\_IMAGES[index]

else:

image = self.default\_image

self.microbit.display.show(image)

def \_setup\_rumble(self):

"""Setup the three animations which simulate a rumble."""

self.left\_rumble = self.\_get\_ready\_to('99500')

self.right\_rumble = self.\_get\_ready\_to('00599')

self.double\_rumble = self.\_get\_ready\_to('99599')

def \_set\_name(self):

self.name = "BBC microbit Gamepad"

def \_set\_evdev\_state(self):

self.\_evdev = False

@staticmethod

def \_get\_target\_function():

return microbit\_process

def \_get\_data(self, read\_size):

"""Get data from the character device."""

return self.\_pipe.recv\_bytes()

def \_get\_ready\_to(self, rumble):

"""Watch us wreck the mike!

PSYCHE!"""

# pylint: disable=no-member

return [self.microbit.Image(':'.join(

[rumble if char == '1' else '00500'

for char in code])) for code in SPIN\_UP\_MOTOR]

def \_full\_speed\_rumble(self, images, duration):

"""Simulate the motors running at full."""

while duration > 0:

self.microbit.display.show(images[0]) # pylint: disable=no-member

time.sleep(0.04)

self.microbit.display.show(images[1]) # pylint: disable=no-member

time.sleep(0.04)

duration -= 0.08

def \_spin\_up(self, images, duration):

"""Simulate the motors getting warmed up."""

total = 0

# pylint: disable=no-member

for image in images:

self.microbit.display.show(image)

time.sleep(0.05)

total += 0.05

if total >= duration:

return

remaining = duration - total

self.\_full\_speed\_rumble(images[-2:], remaining)

self.set\_display()

def set\_vibration(self, left\_motor, right\_motor, duration):

"""Control the speed of both motors seperately or together.

left\_motor and right\_motor arguments require a number:

0 (off) or 1 (full).

duration is miliseconds, e.g. 1000 for a second."""

if left\_motor and right\_motor:

return self.\_spin\_up(self.double\_rumble, duration/1000)

if left\_motor:

return self.\_spin\_up(self.left\_rumble, duration/1000)

if right\_motor:

return self.\_spin\_up(self.right\_rumble, duration/1000)

return -1

def microbit\_process(pipe):

"""Simple subprocess for reading mouse events on the microbit."""

gamepad\_listener = MicroBitListener(pipe)

gamepad\_listener.listen()

class MicroBitListener(BaseListener):

"""Tracks the current state and sends changes to the MicroBitPad

device class."""

def \_\_init\_\_(self, pipe):

super(MicroBitListener, self).\_\_init\_\_(pipe)

self.active = True

self.events = []

self.state = set((

('Absolute', 0x10, 0),

('Absolute', 0x11, 0),

('Key', 0x130, 0),

('Key', 0x131, 0),

('Key', 0x13a, 0),

('Key', 0x133, 0),

('Key', 0x134, 0),

))

self.dpad = True

self.sensitivity = 300

# pylint: disable=import-error

import microbit

self.microbit = microbit

def listen(self):

"""Listen while the device is active."""

while self.active:

self.handle\_input()

def uninstall\_handle\_input(self):

"""Stop listing when active is false."""

self.active = False

def handle\_new\_events(self, events):

"""Add each new events to the event queue."""

for event in events:

self.events.append(

self.create\_event\_object(

event[0],

event[1],

int(event[2])))

def handle\_abs(self):

"""Gets the state as the raw abolute numbers."""

# pylint: disable=no-member

x\_raw = self.microbit.accelerometer.get\_x()

y\_raw = self.microbit.accelerometer.get\_y()

x\_abs = ('Absolute', 0x00, x\_raw)

y\_abs = ('Absolute', 0x01, y\_raw)

return x\_abs, y\_abs

def handle\_dpad(self):

"""Gets the state of the virtual dpad."""

# pylint: disable=no-member

x\_raw = self.microbit.accelerometer.get\_x()

y\_raw = self.microbit.accelerometer.get\_y()

minus\_sens = self.sensitivity \* -1

if x\_raw < minus\_sens:

x\_state = ('Absolute', 0x10, -1)

elif x\_raw > self.sensitivity:

x\_state = ('Absolute', 0x10, 1)

else:

x\_state = ('Absolute', 0x10, 0)

if y\_raw < minus\_sens:

y\_state = ('Absolute', 0x11, -1)

elif y\_raw > self.sensitivity:

y\_state = ('Absolute', 0x11, 1)

else:

y\_state = ('Absolute', 0x11, 1)

return x\_state, y\_state

def check\_state(self):

"""Tracks differences in the device state."""

if self.dpad:

x\_state, y\_state = self.handle\_dpad()

else:

x\_state, y\_state = self.handle\_abs()

# pylint: disable=no-member

new\_state = set((

x\_state,

y\_state,

('Key', 0x130, int(self.microbit.button\_a.is\_pressed())),

('Key', 0x131, int(self.microbit.button\_b.is\_pressed())),

('Key', 0x13a, int(self.microbit.pin0.is\_touched())),

('Key', 0x133, int(self.microbit.pin1.is\_touched())),

('Key', 0x134, int(self.microbit.pin2.is\_touched())),

))

events = new\_state - self.state

self.state = new\_state

return events

def handle\_input(self):

"""Sends differences in the device state to the MicroBitPad

as events."""

difference = self.check\_state()

if not difference:

return

self.events = []

self.handle\_new\_events(difference)

self.update\_timeval()

self.events.append(self.sync\_marker(self.timeval))

self.write\_to\_pipe(self.events)

devices = DeviceManager() # pylint: disable=invalid-name

def get\_key():

"""Get a single keypress from a keyboard."""

try:

keyboard = devices.keyboards[0]

except IndexError:

raise UnpluggedError("No keyboard found.")

return keyboard.read()

def get\_mouse():

"""Get a single movement or click from a mouse."""

try:

mouse = devices.mice[0]

except IndexError:

raise UnpluggedError("No mice found.")

return mouse.read()

def get\_gamepad(blocking=True):

"""Get a single action from a gamepad."""

try:

gamepad = devices.gamepads[0]

except IndexError:

raise UnpluggedError("No gamepad found.")

return gamepad.read(blocking=blocking)

# Код микроконтроллера (приемер для трех кналов PWM)

#include <Servo.h> // подключаем библиотеку для сервопривода

Servo servo\_Yaw; // объявляем переменную servo типа "servo"

Servo servo\_Pith; // объявляем переменную servo типа "servo"

Servo control\_Cam; // объявляем переменную servo типа "servo"

int Yaw\_anle,Pith\_anle,Cam\_mode;

#define PARSE\_AMOUNT 4 // число значений в массиве, который хотим получить

#define INPUT\_AMOUNT 17 // максимальное количество символов в пакете, который идёт в сериал

char inputData[INPUT\_AMOUNT]; // массив входных значений (СИМВОЛЫ)

int intData[PARSE\_AMOUNT]; // массив численных значений после парсинга

boolean recievedFlag;

boolean getStarted;

byte index;

String string\_convert;

void parsing() {

while (Serial.available() > 0) {

char incomingByte = Serial.read(); // ЧИТАЕМ входящий символ

if (incomingByte == '$') { // если это $

getStarted = true; // поднимаем флаг, что можно парсить

} else if (incomingByte != ';' && getStarted) {

inputData[index] = incomingByte;

index++;

inputData[index] = NULL;

} else {

if (getStarted) {

char \*p = inputData;

char \*str;

index = 0;

String value = "";

//while ((str = strtok\_r(p, " ", & p)) != NULL) {

while ((str = strtok\_r(p, ",", & p)) != NULL) {

string\_convert = str;

intData[index] = string\_convert.toInt();

index++;

}

index = 0;

}

}

if (incomingByte == ';') { // ; - конец парсинга

getStarted = false;

recievedFlag = true;

}

}

}

void setup() {

Serial.begin(9600); // подключаем последовательный порт

servo\_Yaw.attach(3); // назначаем канал управления рысканьем на выход 3, влево, вправо

servo\_Pith.attach(5); // назначаем канал управления тангажа на выход 5, вверх, вниз

control\_Cam.attach(9); // назначаем канал управления источником видеосигнала на выход 9

Yaw\_anle = 1500; // инициируем начальное положение сервоприводов в нейтраль

Pith\_anle = 1500; // инициируем начальное положение сервоприводов в нейтраль

Cam\_mode = 1500; // инициируем начальный источник видеосигнала камеру видимого диапазона

servo\_Yaw.writeMicroseconds(Yaw\_anle); // генерируем ШИМ

servo\_Pith.writeMicroseconds(Pith\_anle); // генерируем ШИМ

control\_Cam.writeMicroseconds(Cam\_mode); // генерируем ШИМ

}

void loop()

{

parsing(); // функция парсинга сообщения последовательного порта

if (recievedFlag) { // если получены данные

recievedFlag = false;

Yaw\_anle = intData[0];

//Serial.print(Yaw\_anle);

Pith\_anle = intData[1];

//Serial.print(Pith\_anle);

Cam\_mode = intData[2];

//Serial.print(Cam\_mode);

if (Yaw\_anle > 90){ // ограничиваем углы в соответствии с настройками и возможностями подвеса

Yaw\_anle = 90;

}

if (Yaw\_anle < -90){

Yaw\_anle = -90;

}

if (Pith\_anle > 45){ // ограничиваем углы в соответствии с настройками и возможностями подвеса

Pith\_anle = 45;

}

if (Pith\_anle < -120){

Pith\_anle = -120;

}

if (Cam\_mode == 0){ // формируем ШИМ для переключения источника видеосигнала

Cam\_mode = 1000;

}

if (Cam\_mode == 1){

Cam\_mode = 1500;

}

Yaw\_anle = map(Yaw\_anle, -90, 90, 1000, 2000); // переводим значения углов в параметры ШИМ

Pith\_anle = map(Pith\_anle, -120, 120, 1000, 2000); // переводим значения углов в параметры ШИМ

servo\_Yaw.writeMicroseconds(Yaw\_anle);

servo\_Pith.writeMicroseconds(Pith\_anle);

control\_Cam.writeMicroseconds(Cam\_mode);

}

}