GUI.py documentation that will hopefully be useful to someone.

This file is responsible for the documentation for the main GUI part of the project.

Please insure that any code updates or additions follow object oriented programming principles.

There is plenty left to be done. Such as further object oriented optimizations like a seperate class to handle the commands that will be send to the crawler. As oppsed to having to copy and paste the same lines of code for each function, to name one.

In [ ]:
```
import cv2, tkinter as tk, customtkinter # type: ignore
from PIL import Image, ImageTk # type: ignore
from datetime import datetime
import uuid

import multiprocessing
import socket, pickle
```

Dependencies

Fimiliarize yourself with the following modules/libraries.

Most important modules/libraries:

- CustomTkinter: GUI toolkit.
- Socket and Pickle: For sending and receiving data over web sockets.
- cv2: For capturing video streams.

In [ ]:
```
SERVER_CRAWLER = '192.168.0.19'
CMDPORT = 8000

SERVER_CONTROL_BOX = '192.168.0.23' #change ip in prod
# SERVER_CONTROL_BOX = '192.168.0.26' # Enter CONTROL BOX address
CTRLBXPORT_0 = 10000

server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server.setsockopt(socket.SOL_SOCKET, socket.SO_SNDBUF, 2048)
```

Server Connection

This is the setup for the server. It is a TCP connection.

In [ ]:
```
class App(customtkinter.CTk):

    customtkinter.set_appearance_mode("dark")
    global rec_toggle, video_screen_dim
    rec_toggle = False
    video_screen_dim = (960, 540)

    def __init__(self):
        super().__init__()

        width = self.winfo_screenwidth()
        height = self.winfo_screenheight()
        self.geometry("{}x{}".format(width, height))
        self.title("Control Panel")
        self.wm_iconbitmap(default=None)
        self.maxsize(width=1920, height=height)

        # 20x20 grid system

        self.grid_rowconfigure(tuple(range(10)), weight=1)
        self.grid_columnconfigure(tuple(range(10)), weight=1)

        # logo

        kgb_logo = customtkinter.CTkImage(Image.open("KGB_Logo.png"), size=(160, 75
        logo = customtkinter.CTkLabel(self, text="", image=kgb_logo)
        logo.grid(row=5, column=9, sticky="ne")

        # time display

        self.time = customtkinter.CTkTextbox(master=self,height=10, font=("", 20))
        self.time.grid(row=0, column=0, padx=20, pady=20, sticky="w")
        self.time.insert("0.0", 'CURRENT_TIME')
        self.time_start()

        # window buttons

        self.button = customtkinter.CTkButton(master=self, command=self.max_window,
        self.button.grid(row=0, column=7, padx=(200, 0), pady=20, sticky="e")

        self.button = customtkinter.CTkButton(master=self, command=self.mini_window
        self.button.grid(row=0, column=8, padx=(40, 0), pady=20)

        self.button = customtkinter.CTkButton(master=self, command=self.close_windo
        self.button.grid(row=0, column=9, padx=(0, 20), pady=20, sticky="e")

        # tether buttons

        self.frame = TetherButtonGroup(master=self)
```

```
        self.frame.grid(row=2, column=0, columnspan=1, padx=(20, 0), pady=20, stick

        # movement frame

        self.frame = MovementButtonGroup(master=self)
        self.frame.grid(row=3, column=0, padx=(20, 0), pady=20, sticky="ew")

        # gripper frame

        self.frame = GripperButtonGroup(master=self)
        self.frame.grid(row=4, column=0, columnspan=1, padx=20, pady=20, sticky="w"

        # arm frame

        self.frame = ArmButtonGroup(master=self)
        self.frame.grid(row=5, column=0, padx=(20, 0), pady=20, sticky="w")

        # video buttons

        self.label = customtkinter.CTkLabel(self, text="Video Settings")
        self.label.grid(row=2, column=1, padx=20, pady=(0, 0), sticky="ne")

        self.record_on = customtkinter.CTkButton(master=self, command=self.program_
        self.record_on.grid(row=2, column=1, padx=0, pady=(50,50), sticky="ne")

        self.record_off = customtkinter.CTkButton(master=self, command=self.program
        self.record_off.grid(row=2, column=1, padx=0, pady=(100,0), sticky="ne")

        self.button = customtkinter.CTkButton(master=self, command=self.program_tak
        self.button.grid(row=2, column=1, padx=0, pady=(150, 0), sticky="ne")

        # video device

        self.vid = VideoCaptureDevice()
        self.canvas = tk.Canvas(self, width=0, height=0, bg='#242424', highlightthi
        self.canvas.grid(row=1, column=2, rowspan=4, columnspan=9,padx=20, pady=20,
        self.video_update()

        # info resetter

        self.info_reset()

    def video_update(self):
        try:
            ret, frame = self.vid.get_frame()
            if ret:
                    self.photo = ImageTk.PhotoImage(image= Image.fromarray(frame))
                    self.canvas.create_image(100, 0, image=self.photo, anchor=tk.NW
            self.after(15, self.video_update)
        except Exception as e:
            print(e)

    def program_take_recording(self):
        global rec_toggle
        self.record_on.configure(state='disabled')
        self.vid.get_rec()
```

```
        rec_toggle = True

    def program_stop_recording(self):
        global rec_toggle
        self.record_on.configure(state='enabled')
        rec_toggle = False

    def program_take_picture(self):
        self.vid.get_pic()

    def time_start(self):
        current_time: str = datetime.now().strftime("%H:%M:%S")
        self.time.delete("0.0", "end")
        self.time.insert("0.0", current_time)
        self.after(1000, self.time_start)

    def max_window(self):
        self.geometry("{}x{}".format(1920, 1080))

    def mini_window(self):
        self.geometry("{}x{}".format(300, 300))

    def close_window(self):
        self.destroy()

    def info_reset(self):
        info_to_crawler = {'GRIP': '', 'ARM': ''}
        x_as_bytes = pickle.dumps(info_to_crawler)
        server.sendto((x_as_bytes), (SERVER_CRAWLER, CMDPORT))
        self.after(50, self.info_reset)
```

Main App

This is the main application that will be popping up when you run the script.

Like any other tkinter application, when you defince say a 'button', you must assign a function if you want it to do a command.

Also to note the after() method must be used to run a function asynchronously. A thread will break the program as the app must be in the mainloop. and a new process wont share the memory without a proper handler.

In [ ]:
```
if __name__ == "__main__":
    info_to_control = multiprocessing.Manager().Value('i', {'DIRECTION': ''})
    t = multiprocessing.Process(target=server_listener_start, args=(info_to_control
    app = App()
    t.start()
    app.mainloop()
```

Proper Idiom for multi-processing in Python

For anymore multiprocesses, it is important for the code to be like this for the code to run.

In [ ]:
```
def server_listener_start(info_to_control, server):
    server.bind((SERVER_CONTROL_BOX, CTRLBXPORT_0))
    try:
        while True:
            data_from_crawler = server.recvfrom(2048)
            data_from_crawler = data_from_crawler[0]
            data = pickle.loads(data_from_crawler)
            info_to_control.value = data
    except Exception as e:
        print(e)
```

Server listening

As the functions name suggest, this function listens for data from the crawler. Mainly to see if the data from the GUI ({'CRAWL': 'FORW}), is being recived from the crawler, making the distance counter change. If the distance counter is not changing when the function is called. Then that means there is an issue with the connection or the crawler.

In [ ]:
```
class VideoCaptureDevice:
    #highest res on pi is 1280, 720 using usb
    def __init__(self):
        self.vid = cv2.VideoCapture('http://192.168.0.19:9200/stream.mjpg') #change
        # self.vid = cv2.VideoCapture(None)
        self.rec = None

    def get_frame(self) -> tuple[bool, list[int]]:
        ret, frame = self.vid.read()
        if rec_toggle:
                self.rec.write(frame)
        resized = cv2.resize(frame, video_screen_dim, interpolation=cv2.INTER_AREA)
        return (ret, cv2.cvtColor(resized, cv2.COLOR_BGR2RGB))

    def get_rec(self) -> object:
        unique_id = str(uuid.uuid4()).split('-')[0]
        file_name = f"{unique_id}.avi"
        fourcc = cv2.VideoWriter_fourcc(*'FMP4')
        fps = 10.0
        # res = (640, 480)
        self.rec = cv2.VideoWriter(file_name, fourcc, fps, video_screen_dim)
        return self.rec

    def get_pic(self) -> None:
        ret, frame = self.vid.read()
        if ret:
```

```
            unique_id = str(uuid.uuid4()).split('-')[0] #test code TEST THIS
            cv2.imwrite(f"{unique_id}.png", frame)

    def __del__(self) -> None:
        if self.vid.isOpened():
            try:
                self.vid.release()
                self.rec.release()
            except Exception as e:
                print(e)
```

Video processing

This class handles the video processing. It reads the video from the camera, resizes it to the screen size, and writes it to a file if recording is enabled. It also handles taking pictures when the picture button is pressed. The image is saved with a unique identifier using an external library, UUID.

In [ ]:
```
class TetherButtonGroup(customtkinter.CTkFrame):

    TETH_MTR = Motor(13, 6)

    def __init__(self, master):
        super().__init__(master)

        self.extend = 0 # test code REMOVE

        # tether buttons
        self.grid_rowconfigure(tuple(range(9)), weight=1)
        self.grid_columnconfigure(tuple(range(9)), weight=1)

        self.label = customtkinter.CTkLabel(self, text="Tether")
        self.label.grid(row=0, column=0, pady=20)

        self.button = customtkinter.CTkButton(master=self, command=self.tether_exte
        self.button.grid(row=1, column=0, padx=20, pady=20)

        self.button = customtkinter.CTkButton(master=self, command=self.tether_stop
        self.button.grid(row=1, column=1, padx=20, pady=20)

        self.button = customtkinter.CTkButton(master=self, command=self.tether_retr
        self.button.grid(row=1, column=2, padx=20, pady=20)

    def tether_extend(self):
        self.TETH_MTR.forward()

    def tether_stop(self):
        self.TETH_MTR.stop()

    def tether_retract(self):
        self.TETH_MTR.backward()
```

Tether Button Group

This is a frame from customtkinter that contains the tether buttons for extending, stopping, and retracting the tether manually.

In [ ]:
```
class MovementButtonGroup(customtkinter.CTkFrame):

    meters = 0

    def __init__(self, master):
        super().__init__(master)

        # movement buttons

        self.grid_rowconfigure(tuple(range(9)), weight=1)
        self.grid_columnconfigure(tuple(range(9)), weight=1)

        self.label = customtkinter.CTkLabel(self, text="Movement")
        self.label.grid(row=0, column=0, pady=20)

        self.button = customtkinter.CTkButton(master=self, command=self.crawler_for
        self.button.grid(row=1, column=1, padx=5, pady=(5, 0))

        self.button = customtkinter.CTkButton(master=self, command=self.crawler_rig
        self.button.grid(row=2, column=2, padx=5, pady=5)

        self.button = customtkinter.CTkButton(master=self, command=self.crawler_bac
        self.button.grid(row=3, column=1, padx=5, pady=(5, 20))

        self.button = customtkinter.CTkButton(master=self, command=self.crawler_lef
        self.button.grid(row=2, column=0, padx=5, pady=5)

        self.button = customtkinter.CTkButton(master=self, command=self.crawler_sto
        self.button.grid(row=2, column=1, padx=5, pady=5, sticky='n')

        self.button = customtkinter.CTkButton(master=self, command=self.crawler_shu
        self.button.grid(row=2, column=3, padx=20, pady=5)

        # estimated distance

        self.label = customtkinter.CTkLabel(self, text="Distance")
        self.label.grid(row=0, column=3, pady=20, sticky='w')

        self.distance = customtkinter.CTkTextbox(master=self,height=10, font=("", 2
        self.distance.grid(row=1, column=3, padx=(5,0), pady=5)
        self.distance.insert("0.0", "0 m")
        self.position_change()

    def crawler_forward(self):
        info_to_crawler = {'CRAWL': 'FORW'}
        x_as_bytes = pickle.dumps(info_to_crawler)
        server.sendto((x_as_bytes), (SERVER_CRAWLER, CMDPORT))
```

```
    def crawler_backward(self):
        info_to_crawler = {'CRAWL': 'BACK'}
        x_as_bytes = pickle.dumps(info_to_crawler)
        server.sendto((x_as_bytes), (SERVER_CRAWLER, CMDPORT))

    def crawler_right(self):
        info_to_crawler = {'CRAWL': 'RIGHT'}
        x_as_bytes = pickle.dumps(info_to_crawler)
        server.sendto((x_as_bytes), (SERVER_CRAWLER, CMDPORT))

    def crawler_left(self):
        info_to_crawler = {'CRAWL': 'LEFT'}
        x_as_bytes = pickle.dumps(info_to_crawler)
        server.sendto((x_as_bytes), (SERVER_CRAWLER, CMDPORT))

    def crawler_stop(self):
        info_to_crawler = {'CRAWL': 'STOP'}
        x_as_bytes = pickle.dumps(info_to_crawler)
        server.sendto((x_as_bytes), (SERVER_CRAWLER, CMDPORT))

    def crawler_shutdown(self):
        info_to_crawler = {'CRAWL': 'SHUTDOWN'}
        x_as_bytes = pickle.dumps(info_to_crawler)
        server.sendto((x_as_bytes), (SERVER_CRAWLER, CMDPORT))

    def position_change(self):
        if info_to_control.value['DIRECTION'] == 'FORW':
            self.meters += 0.01
        elif info_to_control.value['DIRECTION'] == 'BACK':
            self.meters -= 0.01
        self.distance.delete("0.0", "end")
        self.distance.insert("0.0", f'{round(self.meters, 2)} m')
        self.after(500, self.position_change)
```

Movement Button Group

This is another frame from customtkinter that contains the movement buttons for controlling the robot's movement. It also keeps track of the distance traveled by the robot roughly.

In [ ]:

```
class GripperButtonGroup(customtkinter.CTkFrame):
    def __init__(self, master):
        super().__init__(master)

        self.label = customtkinter.CTkLabel(self, text="Claw")
        self.label.grid(row=0, column=0, pady=20)

        # gripper buttons

        self.button = customtkinter.CTkButton(master=self, command=self.gripper_ope
        self.button.grid(row=1, column=0, padx=20, pady=20)

        self.button = customtkinter.CTkButton(master=self, command=self.gripper_clo
```

```
        self.button.grid(row=1, column=1, padx=20, pady=20)

        self.button = customtkinter.CTkButton(master=self, command=self.gripper_rig
        self.button.grid(row=1, column=2, padx=20, pady=20)

        self.button = customtkinter.CTkButton(master=self, command=self.gripper_lef
        self.button.grid(row=1, column=3, padx=20, pady=20)

    def gripper_open(self):
        info_to_crawler = {'GRIP': 'OPEN'}
        x_as_bytes = pickle.dumps(info_to_crawler)
        server.sendto((x_as_bytes), (SERVER_CRAWLER, CMDPORT))

    def gripper_close(self):
        info_to_crawler = {'GRIP': 'CLOSE'}
        x_as_bytes = pickle.dumps(info_to_crawler)
        server.sendto((x_as_bytes), (SERVER_CRAWLER, CMDPORT))

    def gripper_left(self):
        info_to_crawler = {'GRIP': 'LEFT'}
        x_as_bytes = pickle.dumps(info_to_crawler)
        server.sendto((x_as_bytes), (SERVER_CRAWLER, CMDPORT))

    def gripper_right(self):
        info_to_crawler = {'GRIP': 'RIGHT'}
        x_as_bytes = pickle.dumps(info_to_crawler)
        server.sendto((x_as_bytes), (SERVER_CRAWLER, CMDPORT))
```

Gripper/Hand Button Group

This is another frame from customtkinter that contains the gripper and hand buttons for controlling the robot's hand.

In [ ]:

```
class ArmButtonGroup(customtkinter.CTkFrame):
    def __init__(self, master):
        super().__init__(master)

        self.label = customtkinter.CTkLabel(self, text="Arm")
        self.label.grid(row=0, column=0, pady=20)

        # arm buttons

        self.button = customtkinter.CTkButton(master=self, command=self.arm_extend,
        self.button.grid(row=1, column=0, padx=20, pady=20)

        self.button = customtkinter.CTkButton(master=self, command=self.arm_retract
        self.button.grid(row=1, column=1, padx=20, pady=20)

    def arm_extend(self):
        info_to_crawler = {'ARM': 'EXT'}
        x_as_bytes = pickle.dumps(info_to_crawler)
        server.sendto((x_as_bytes), (SERVER_CRAWLER, CMDPORT))
```

```
    def arm_retract(self):
        info_to_crawler = {'ARM': 'RETR'}
        x_as_bytes = pickle.dumps(info_to_crawler)
        server.sendto((x_as_bytes), (SERVER_CRAWLER, CMDPORT))
```

Arm Button Group

This is another frame from customtkinter that contains the arm buttons for controlling the robot's arm.