

LILIT-2026_plieuse

Alain.Empain@uliege.be

CheapYellowDisplay pour piloter une plieuse de plastic

Trois frames de menu

Setup de travail :

type de plastic (plexi, PVC, polyéthylène...)

épaisseur,

température ambiante pour correction de temps

Action :

start (start/stop automatiques)

stop (pour interruption en cours de cycle)

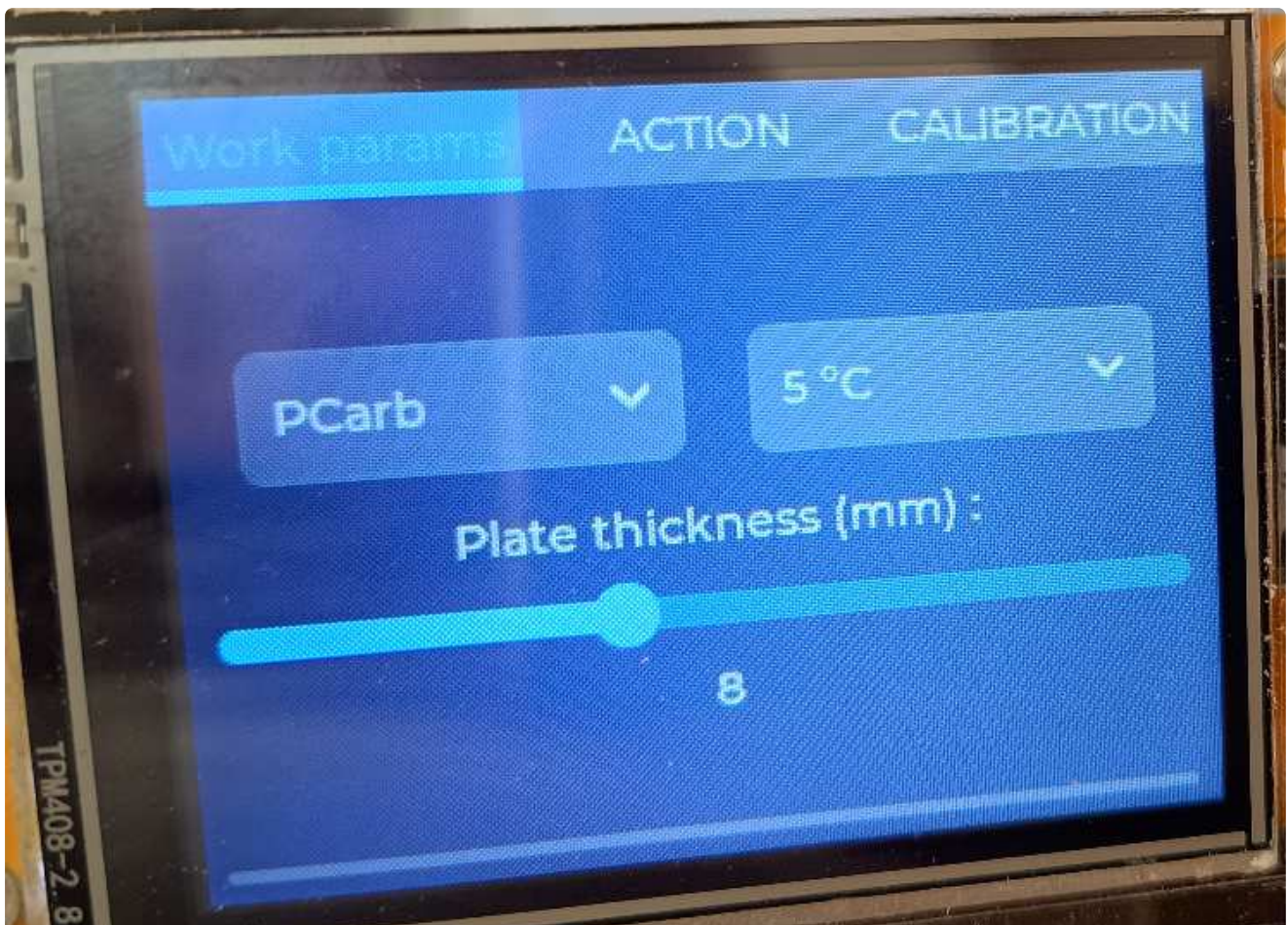
Calibrages du système :

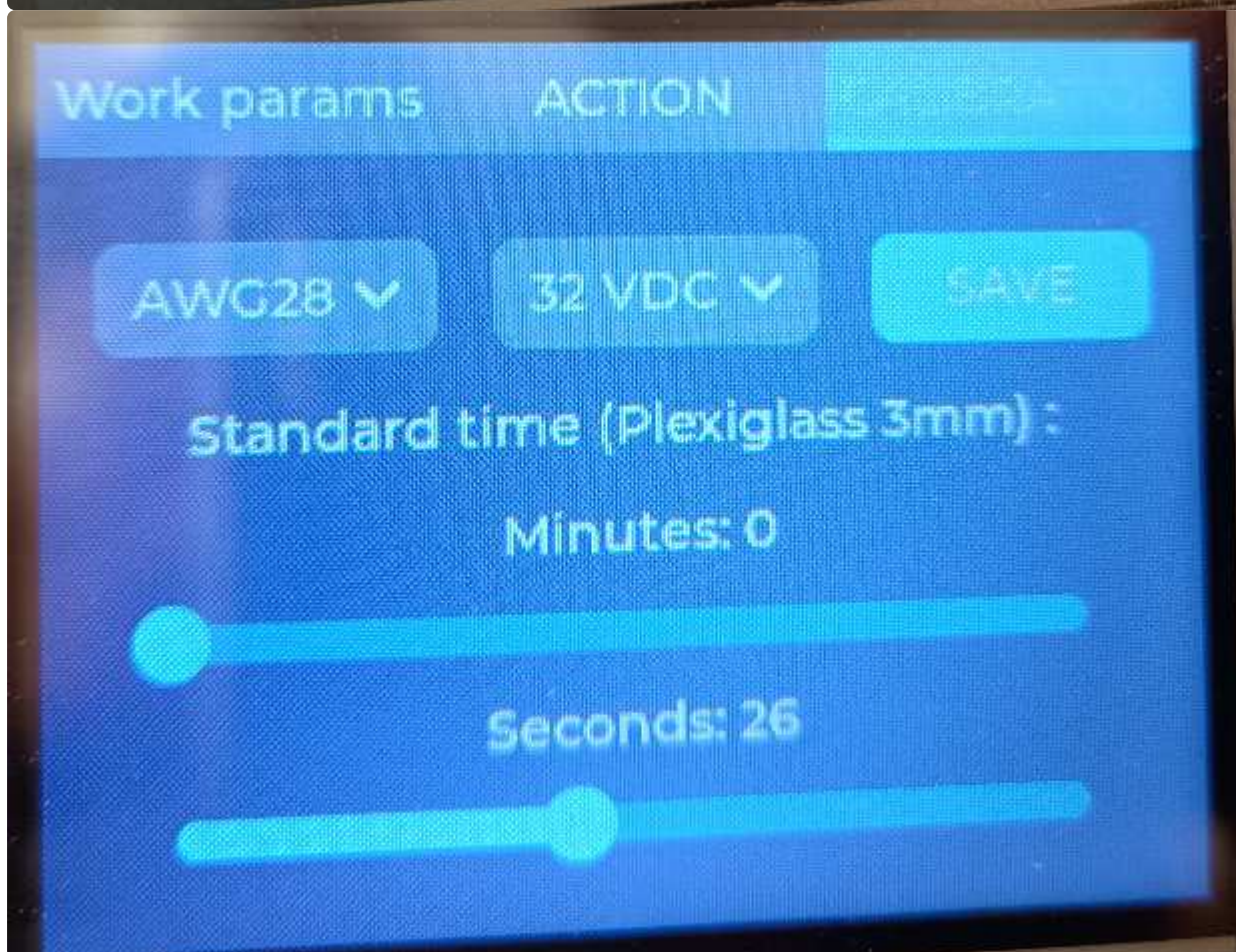
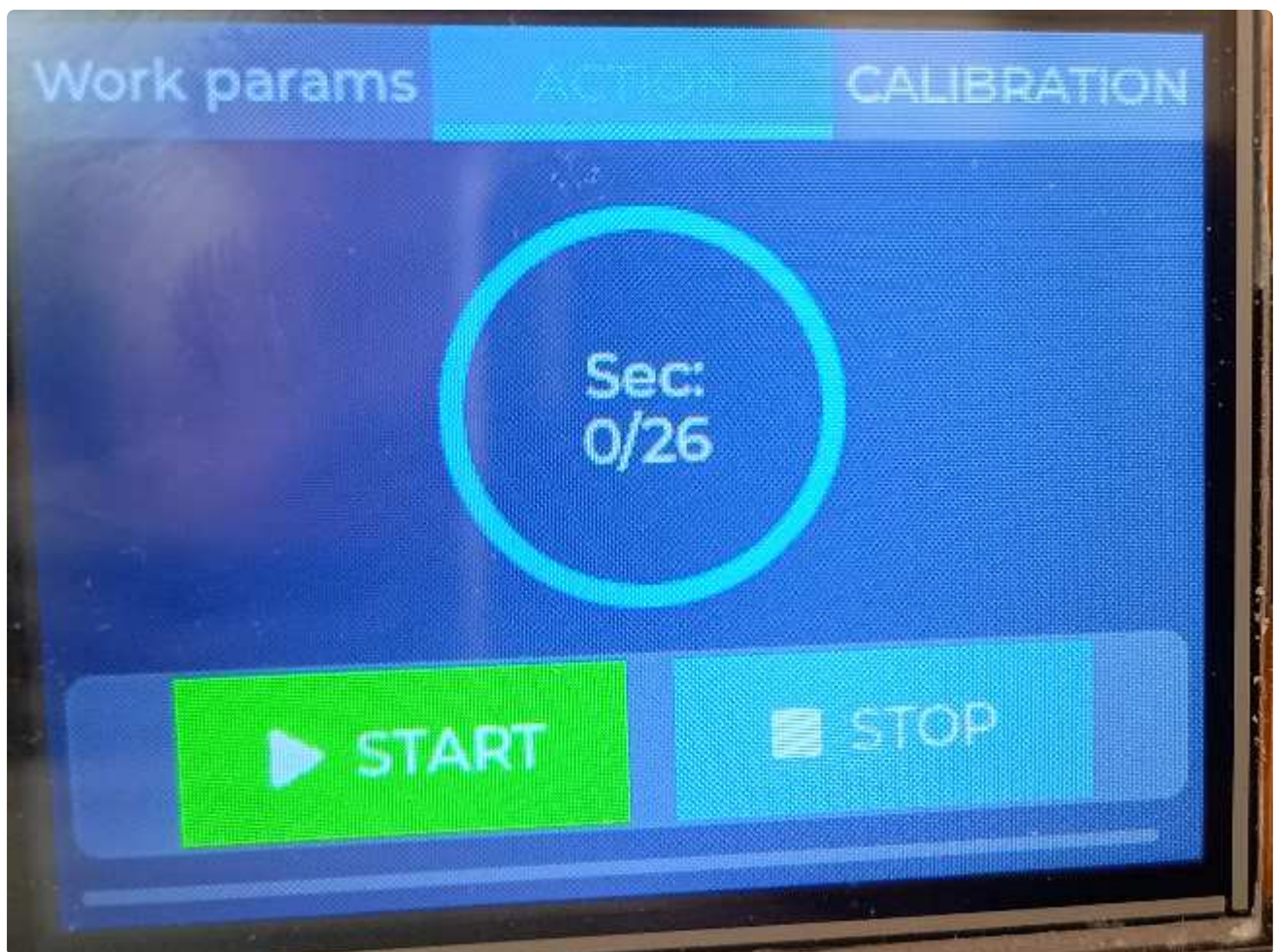
épaisseur du fil NiChrome

voltage de la source

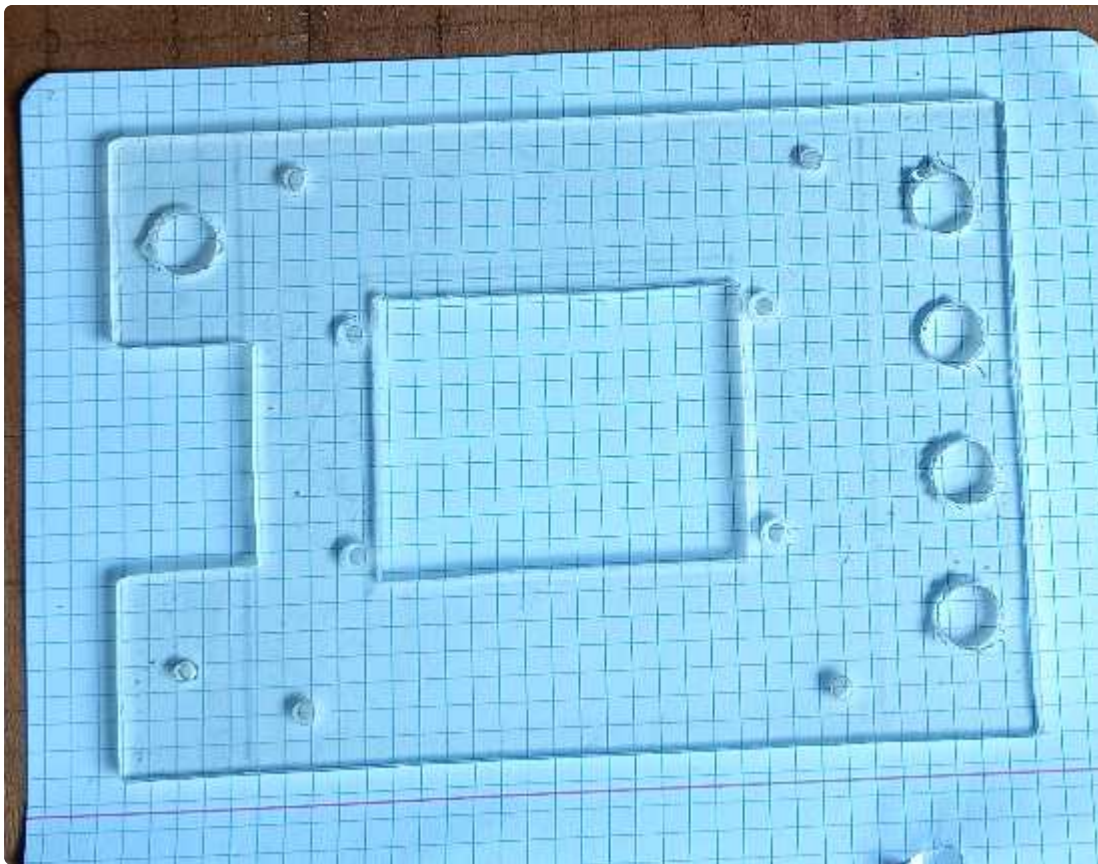
temps standard pour plier 3mm de plexi (min et secondes)

[SAVE] sauve le tout dans un fichier au format JSON sur le stockage permanent (flash disk), rechargé automatiquement au démarrage.

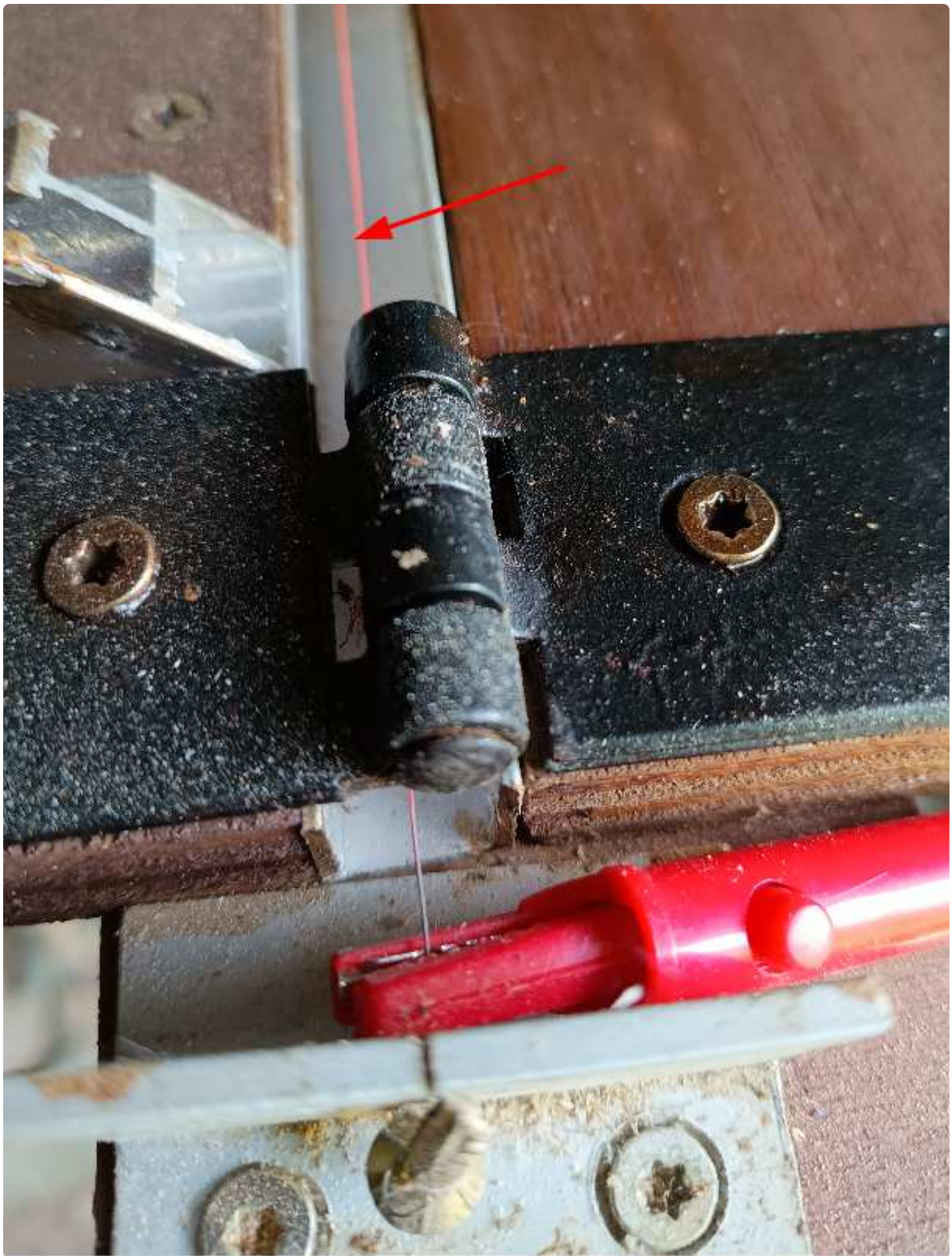




Exemple de réalisation



Plieuse : deux plaques de bois articulées avec charnière, une cornière en U avec un fil en nickel-chrome tendu dans la gorge du U



Je place la plaque de plexi le long du guide, en alignant le trait à plier sur le fil

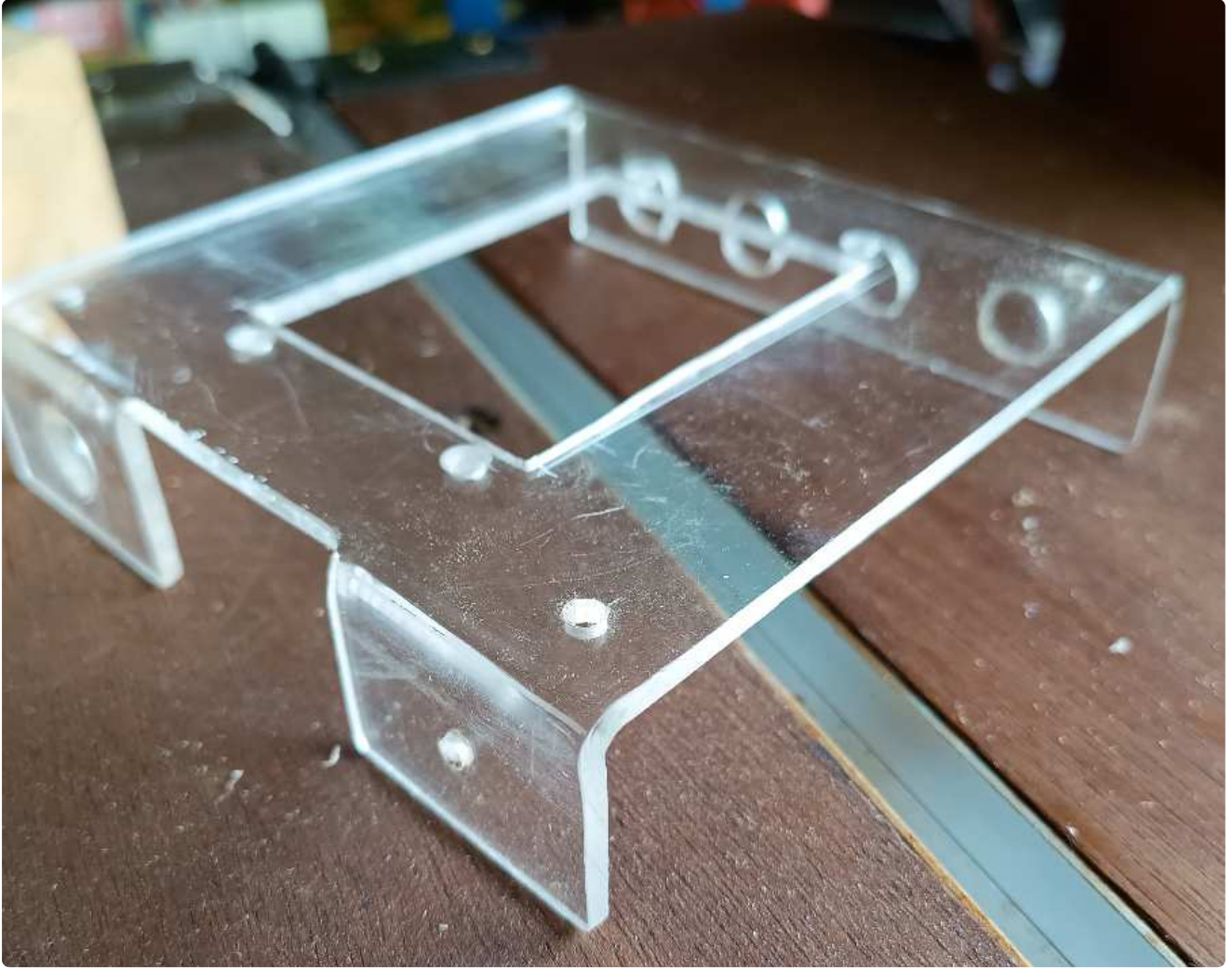


J'attends le temps nécessaire (2min15 pour 3mm de plexi), et je plie grâce aux charnières, et tenant bien en place le côté fixe du plexi.
Comme la partie chauffée est très bien localisée, le pli est impeccablement aligné !

Bien maintenir l'angle voulu pendant le refroidissement (~ une minute) : un simple bloc de bois fait l'affaire.



Et voilà ! Il ne reste plus qu'à y visser le CheapYellowDisplay de la démo.



HARDWARE

1. un ESP32 classique monté avec un écran tactile : **Cheap Yellow DISPLAY**
2. un PC avec Linux pour le développement, l'installation de l'application, le debugging :
connection par USB
(inutile quand l'application est installée en auto-exécution)
3. un relais solid state SSR
4. une alimentation pour le 32 VDC, alimentée en 230VAC par le relais
5. la plieuse : un simple fil de NickelChrome de 0.29 mm de diamètre

PROGRAMMATION

Intro : classes, méthodes

initialisation d'un objet, membre de la classe des mammifères

mammifère(poids, cri, nourriture)

chien = mammifère('1.5', 'woua-woua', 'croquettes')

Extraction des informations

chien.pattes -> 4

chien.reproduction -> vivipare

chien.lactation -> TRUE

chien.cri -> 'woua-woua'

Exécution d'une méthode

chien.nourrir() -> ouvre la boîte aux croquettes

chien.poids(2.1) -> mise-à-jour de son poids

AUTRE EXEMPLE

Initialisation de la classe des carrés : `carré(longueur)`

monCarré = `carré(10)`

Extraction de paramètres

monCarré.côtés -> 4

monCarré.longueur -> 10 `#cm`

Exécution d'une méthode commune à tous les membres de la classe des carrés

monCarré.surface() -> 100 `#cm²`

monCarré.périmètre() -> 40

Configuration

Stockée dans le système de fichiers installé sur la mémoire flash du ESP32 (folder.json)

1- Outils lecture/écriture du fichier JSON

```
import json

#-----
def saveJson(obj,fname) :
    with open(fname,'w') as file :
        json.dump(obj,file)

#-----
def readJson(fname) :
    with open(fname, 'r') as file:
        return(json.load(file))

config={
    "kind": "PAcryl",
    "thickness": "5",

    "nicr": "AWG26",
    "volt": "32",
    "ambient": "20",

    "StdTime": "10",
```

```

    "RAcryl": "1.0",
    "RPVC": "1.0",
    "RPEth": "1.0",
    "RPProp": "1.0",
    "RPCarb": "1.0",
    "RPStyr": "1.0"
}

```

2- Démarrage : initialisation de la config :

- lecture du fichier
- ou création à partir de la config par défaut en cas d'absence du fichier

```

try:
    with open('folder.json') as js :
        config=json.load(js)
except:
    print('<E> no folder.json found, creating it with defaults')
    saveJson(config,"folder.json")

```

Mise en page : trois panneaux sur onglet

```

#=====
#:  Main Interface
#=====

#: creation du container principal, 'tabview'
#:      qui contiendra les trois panneaux

tabview = lv.tabview(scr)

tabview.set_size(320, 240)
tabview.set_tab_bar_size(30);

#: insertion des trois objets-panneaux dans le container
#:      définition des titres des onglets

tab1 = tabview.add_tab("Work params")
tab2 = tabview.add_tab("ACTION")
tab3 = tabview.add_tab("CALIBRATION")

#: activation des trois panneaux ( config, action, param )

config_create(tab1)
action_create(tab2)

```

```
param_create(tab3)
```

Définition des objets de chaque panneau

Widgets utilisés

- boutons
- curseurs
- listes pop-down

Définition des propriétés de chaque widget

- container (parent)
- couleur
- fonte
- texte,
- taille
- position
- ...
- action (call-back) ...

Exemple : panneau de configuration 'config'

et creation de la liste drop-down pour choisir la matière

```
def config_create(parent):

    global dd,dt,dn,dv,slider,tlst
    parent.set_flex_flow(lv.FLEX_FLOW.ROW_WRAP)
    parent.set_flex_align(lv.
        FLEX_ALIGN.SPACE_EVENLY,
        lv.FLEX_ALIGN.CENTER,
        lv.FLEX_ALIGN.CENTER)

    # material -----

    dm = lv.dropdown(parent)
    dm.set_text("Material")
    mlst=("PAcryl","PCarb","PEth","PProp","PStyr","PVC")
    dm.set_options("\n".join(mlst))
    dm.set_size(130, 40)
    dm.add_event_cb(lambda e: kind_cb(dm, dm.get_selected(),
        mlst), lv.EVENT.VALUE_CHANGED, None)
```

Définition d'un 'call-back' en cas d'événement

(clic sur un bouton, déplacement d'un curseur)

la méthode 'add_event_cb' permet de coupler une fonction 'call-back' à un événement

```
dm.add_event_cb(lambda e: kind_cb(dm, dm.get_selected(),
                                  mlst), lv.EVENT.VALUE_CHANGED, None)
```

Ex: couplage de la fonction call-back

dans l'exemple la fonction '**kind_cb**' va être appelée avec ses arguments ad hoc en cas de l'événement 'VALUE_CHANGED' de l'objet dm

(dm= d: data, m: material)

Ex: Fonction call-back 'kind_cb'

```
def kind_cb(obj, idx, lst) :
    knd=lst[idx]
    config['kind']=knd
    obj.set_text(knd)

    print(f"Kind: idx={idx} type:{type(lst)}, {lst}: {knd}\n{config}")
```