

---

## Documentation

last modified by Forestier Robin

on 2022/04/05 10:48

---

# Table of Contents

1.0 - Description .....	4
1.1 - Objectifs .....	4
1.2 - Objectifs globaux .....	4
2.0 - Fonctionnement .....	6
2.1 - Caméras .....	6
2.2 - Serveur .....	8
2.2.1 - Flask .....	8
2.2.2 - Interface utilisateurs .....	8
3.0 - Matériel .....	12
3.1 - Raspberry Pi caméra .....	12
3.2 - Serveur .....	12
4.0 - Installation .....	13
4.1 - Raspberry Pi caméra .....	13
4.2 - Raspberry Pi serveur .....	14
4.3 - Serveur NTP .....	14
5.0 - Programmation .....	15
5.1 - Caméra .....	15
5.1.1 - Sélection des portes .....	15
5.1.2 - Détection des personnes .....	17
5.1.3 - Tracking des personnes .....	18
5.1.4 - Détection des entrées et des sorties .....	21
5.1.5 - Mesures .....	23
5.1.6 - Tests et améliorations .....	23
5.2 - Amélioration réalisée .....	23
5.2.1 - Détection des personnes .....	23
5.2.2 - Tracking des personnes .....	24
5.2.3 - Détection des entrées et des sorties .....	24
5.2.4 - Tests et améliorations .....	24
5.2.5 - Bugs .....	24
5.3 - Serveur .....	25
5.3.1 - Fonctions déjà implémentées (V 1.2) .....	25
5.3.2 - Fonctions à implémenter pour le TPI .....	25
5.3.3 - Page settings .....	25
5.3.4 - Page login 2 .....	25
5.3.5 - Plage horaire .....	27
5.3.6 - Ajout d'une heure .....	29
5.3.7 - Mode manuel .....	30
5.3.8 - Heure manuelle .....	30
5.3.9 - Tâche en arrière plan .....	31
5.4- Communication Wifi .....	31
6.0 - Remerciements .....	33
7.0 - Conclusion .....	33
8.0 - Bugs connus et améliorations .....	33

- [1.0 - Description](#)
  - [1.1 - Objectifs](#)
  - [1.2 - Objectifs globaux](#)
- [2.0 - Fonctionnement](#)
  - [2.1 - Caméras](#)
  - [2.2 - Serveur](#)
    - [2.2.1 - Flask](#)
    - [2.2.2 - Interface utilisateurs](#)
- [3.0 - Matériel](#)
  - [3.1 - Raspberry Pi caméra](#)
  - [3.2 - Serveur](#)
- [4.0 - Installation](#)
  - [4.1 - Raspberry Pi caméra](#)
  - [4.2 - Raspberry Pi serveur](#)
  - [4.3 - Serveur NTP](#)
- [5.0 - Programmation](#)
  - [5.1 - Caméra](#)
    - [5.1.1 - Sélection des portes](#)
    - [5.1.2 - Détection des personnes](#)
    - [5.1.3 - Tracking des personnes](#)
    - [5.1.4 - Détection des entrées et des sorties](#)
    - [5.1.5 - Mesures](#)
    - [5.1.6 - Tests et améliorations](#)
  - [5.2 - Amélioration réalisée](#)
    - [5.2.1 - Détection des personnes](#)
    - [5.2.2 - Tracking des personnes](#)
    - [5.2.3 - Détection des entrées et des sorties](#)
    - [5.2.4 - Tests et améliorations](#)
    - [5.2.5 - Bugs](#)
  - [5.3 - Serveur](#)
    - [5.3.1 - Fonctions déjà implémentées \(V 1.2\)](#)
    - [5.3.2 - Fonctions à implémenter pour le TPI](#)
    - [5.3.3 - Page settings](#)
    - [5.3.4 - Page login 2](#)
    - [5.3.5 - Plage horaire](#)
    - [5.3.6 - Ajout d'une heure](#)
    - [5.3.7 - Mode manuel](#)
    - [5.3.8 - Heure manuelle](#)
    - [5.3.9 - Tâche en arrière plan](#)
  - [5.4- Communication Wifi](#)
- [6.0 - Remerciements](#)
- [7.0 - Conclusion](#)
- [8.0 - Bugs connus et améliorations](#)

# 1.0 - Description

L'objectif du projet est de réaliser un système autonome de contrôle des lumières de notre atelier. Une interface graphique a été développée en amont de ce TPI. Elle sera utilisée comme base. Pour automatiser l'enclenchement et l'extinction des lumières, des caméras doivent être utilisées.

Énoncé du TPI: [Énoncé du travail d'examen](#)  
serveur: **OMVSERVER\formation\TPI\TPI\_Forestier\_Gestion\_Lumiere\_KNX**  
projet sur gitlab: [TPI Forestier Gestion Lumière KNX](#)  
Planning: [Planning](#)

## 1.1 - Objectifs

1. Création de la page settings.
2. Modification des mots de passe.
3. Gestion de la date et de l'heure.
4. Gestion des plages horaires.
5. Mode manuel du système.

## 1.2 - Objectifs globaux

- Utilisation d'un Raspberry Pi et d'une Raspberry Pi camera.
- Gestion des plages horaires sous forme graphique.

**Intitulé du travail****Gestion système lumière KNX - EIB****1. Description du travail d'examen**

Le plan d'action de la formation approfondie choisie dans le plan de formation sert de base pour la description du travail d'examen.

**Avant TPI**

Interface via écran tactile des commandes lumière KNX-EIB de notre atelier.

Possibilité de prendre la main sur l'installation via ethernet.

Gestion caméra via raspberry pi

**TPI**

Depuis l'écran tactile ou remote, accès à la paramétrisation du système protégé par mot de passe.

Depuis cette page :

On peut changer de mot de passe

Gestion de la date et heure : manuellement et/ou via serveur NTP ( ch.pool.ntp.org )

Gestion par plage horaire hebdomadaire :

des extinctions automatiques ( lumières séparées ou groupées )

définition des plages pour valider les caméras ( caméra peuvent enclencher ou déclencher les lumières )

gestion des lumières enclenchées via caméra (selon plan hebdomadaire)

Passage en mode manuel ( quel que soit la plage horaire ). Cette commande est annulée soit manuellement soit automatiquement par le passage à minuit

**2. Exigences particulières, infrastructure, informations complémentaires**

Un raspberry pi par caméra.

Gestion des plages horaires sous formes graphiques

Contrôle des caméras ( en fonctionnement ou erreur )

La caméra enclenche la lumière sans délai. Plus de présence détectée , déclenchement après 10 minutes

**3. Informations complémentaires**

Dans l'entreprise, le travail est

☒

unique

☐

répétitif

Le candidat exécute le travail pour

☒

la 1<sup>er</sup> fois

☐

la 2<sup>e</sup> ou x<sup>e</sup> fois

☒

Travail individuel.

☐

Travail d'équipe.

**4. Annexes**

☒ Feuille d'appréciation avec les critères d'évaluation discutée avec le candidat

☒ Calendrier approximatif

Autres documents techniques :

- Annexe descriptif réseau et plan horaire ( exemple )

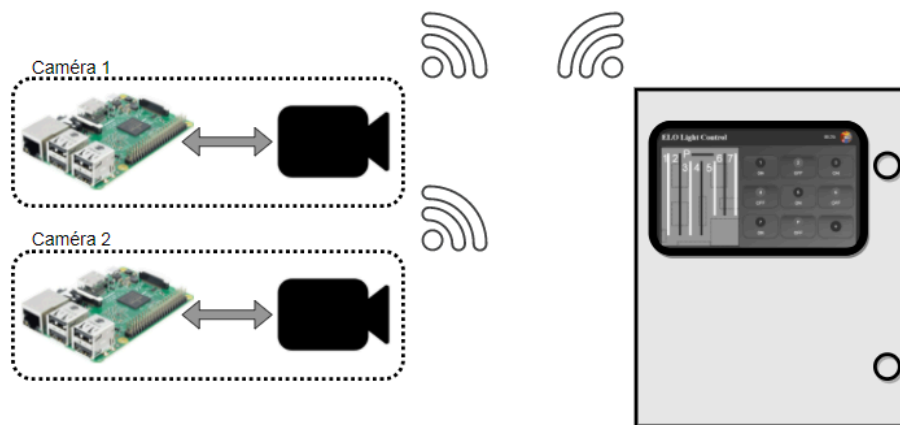
-

-

-

## 2.0 - Fonctionnement

Pour réaliser mon projet, je vais tout au long de celui-ci le séparer en deux parties distinctes. Une partie caméra, servant uniquement à la détection des personnes et une deuxième partie serveur, qui elle, comporte uniquement le serveur Web et l'écran tactile pour le contrôle manuel et l'accès aux réglages.



Chacune des parties dispose de son propre Raspberry Pi 3B+. Pour la partie caméra, chaque Raspberry Pi a sa propre caméra.

Pendant le développement de mon projet, je vais porter une attention particulière à ce que le nombre de Raspberry Pi caméra puisse changer sans poser de problèmes au bon fonctionnement du système. Le système doit aussi fonctionner si la communication échoue.

Chacune des caméras communiquera directement avec le serveur via Wi-Fi, plus précisément via request HTTP.

### 2.1 - Caméras

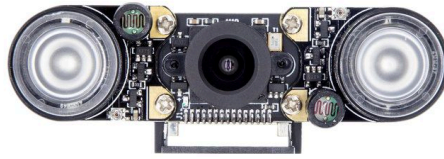
Premièrement, pourquoi utiliser des caméras ?

*Les caméras sont chères, la détection d'objets sans intelligence artificielle est très compliquée et très gourmande en ressources pour un si petit ordinateur qu'est un Raspberry Pi !*

Ces trois affirmations sont parfaitement correctes, mais laissez-moi vous expliquer ce choix. Durant ma troisième année d'apprentissage, j'ai eu la chance de pouvoir choisir un perfectionnement, j'ai donc choisi **le traitement d'images et la détection dans les images**.

Deuxièmement, au vu de la topologie de notre atelier, l'utilisation de capteurs de mouvement est impossible, ceux-ci ne peuvent pas assurer une détection si précise à une distance aussi grande. Le risque qu'une personne finisse dans la nuit est trop élevé.

C'est donc pour cela que la détection des personnes se fera à l'aide de Raspberry Pi 3 avec chacun une Raspberry Pi caméra NoIR avec led infrarouge intégrée de chez PiSupply. Cette caméra se branche directement sur le port CSI-2 Caméra du Raspberry Pi et elle apporte aussi une vision nocturne car le capteur **NoIR** ne filtre pas les rayonnements infrarouges. Grâce aux deux LDR et aux deux LED infrarouges, la caméra gère automatiquement la luminosité des led pour assurer un bon éclairage. La vision nocturne est fonctionnelle uniquement sur une courte distance, moins de 5 mètres environ.

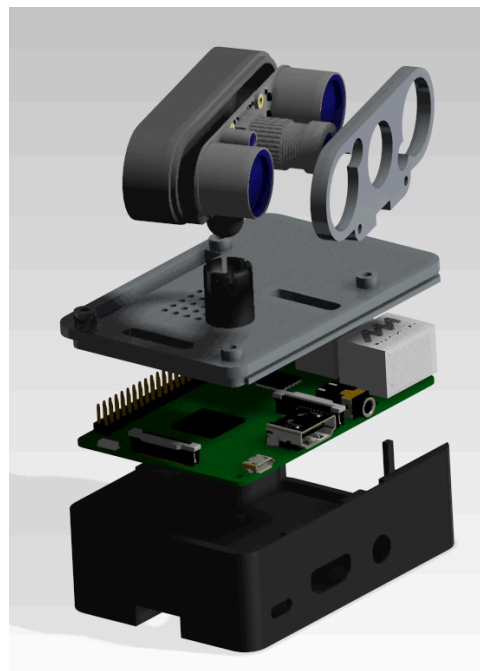


Pour contrôler l'éclairage automatiquement et enclencher les lumières à l'arrivée d'une personne au sein de l'atelier, je dois pouvoir détecter cette personne puis communiquer au serveur pour lui demander d'allumer les lumières. Cette détection devrait, dans le meilleur des cas, fonctionner à toutes heures du jour et de la nuit. Pour cela, j'ai choisi de commander et de tester des caméras comportant des led infrarouges. J'ai pu remarquer que ces led étaient insuffisantes pour observer le moindre mouvement à plus de 5 mètres. Pour la première partie du projet, j'ai conclu avec Monsieur Dupertuis qu'il n'était pas important que la détection soit fonctionnelle de nuit. Mais je garde cette idée comme amélioration possible de mon système.

Pourquoi plusieurs caméras ?

Car nous avons trois portes qui peuvent être utilisées comme entrées. Malheureusement, elles ne peuvent pas être vues les trois en même temps par une seule caméra. Il m'a donc fallu placer plusieurs caméras. Pour faciliter l'utilisation du système, les programmes seront identiques sur tous les Raspberry Pi caméra.

J'ai réalisé avant le TPI un boîtier en 3D pour le Raspberry Pi et pour la caméra. Ce boîtier peut s'empiler si l'on souhaite placer les caméras au même endroit. Ce boîtier est bien, facile à monter et à imprimer en 3D, mais la circulation d'air est très restreinte ce qui fait rapidement augmenter la température du Raspberry Pi. Une deuxième version incorporant un ventilateur serait une bonne idée.



Les fichiers 3D se trouvent dans le GitLab sous: [/4\\_Fabrication/Version\\_01](#).

## 2.2 - Serveur

### 2.2.1 - Flask

Avant mon TPI, j'ai dû réaliser une interface graphique pour le contrôle des lumières de l'atelier.

J'ai pensé au début partir sur une application simple avec une interface simple comme par exemple avec TKinter.

Mais je souhaitais que l'on puisse contrôler les lumières depuis n'importe quels appareils. Pour ce faire, j'ai commencé le développement d'une interface WEB.

J'ai choisi d'utiliser Flask (<https://flask.palletsprojects.com>), qui est un micro framework open-source de développement WEB en python, pour créer mon site. Flask est très facile à maitre en oeuvre, il est recommandé par python et il est très sécurisé.

Mais Flask est uniquement un serveur de développement. Il me fallait un serveur WSGI (Web Server Gateway Interface), j'ai choisi d'utiliser Gunicorn (<https://gunicorn.org>) car il est facile à installer, très léger pour le système et rapide.

Avec cela, j'ai aussi souhaité implémenter un proxy pour permettre une connexion sécurisée en HTTPS. J'ai choisi NGINX (<https://www.nginx.com>), open source, facile à installer et très performant.

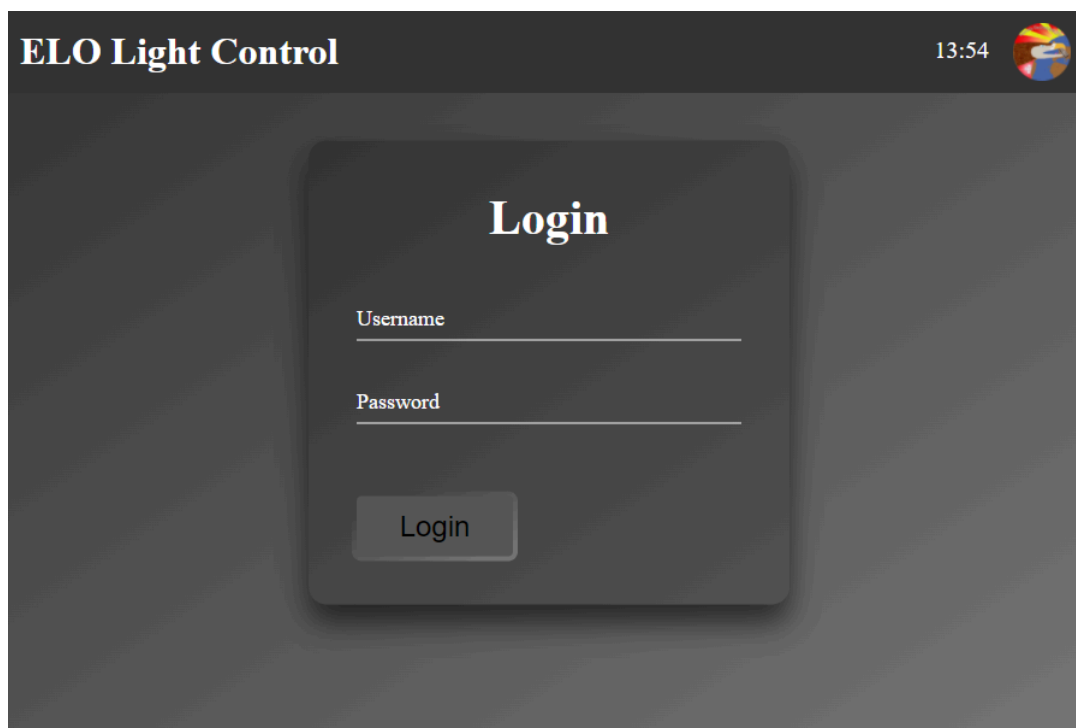
Toute l'installation est détaillée dans le XWiki : <https://xwiki.serverelo.org/xwiki/bin/view/Centre%20de%20Formation%20ELO/Projets/Controle%20des%20lumières%20KNX/>

### 2.2.2 - Interface utilisateurs

Pour se rendre sur le site, il faut s'y connecter via un navigateur internet et en tapant l'adresse IP du Raspberry Pi serveur : **172.16.32.133** (dans mon cas).

Cette action peut uniquement être réalisée si l'on est connecté sur le Wi-Fi de l'atelier ELO ou si on possède l'accès au VPN.

L'utilisateur arrivera sur la page de login, il doit rentrer l'identifiant et le mot de passe.



Il existe trois identifiants avec chacun un mot de passe respectif:

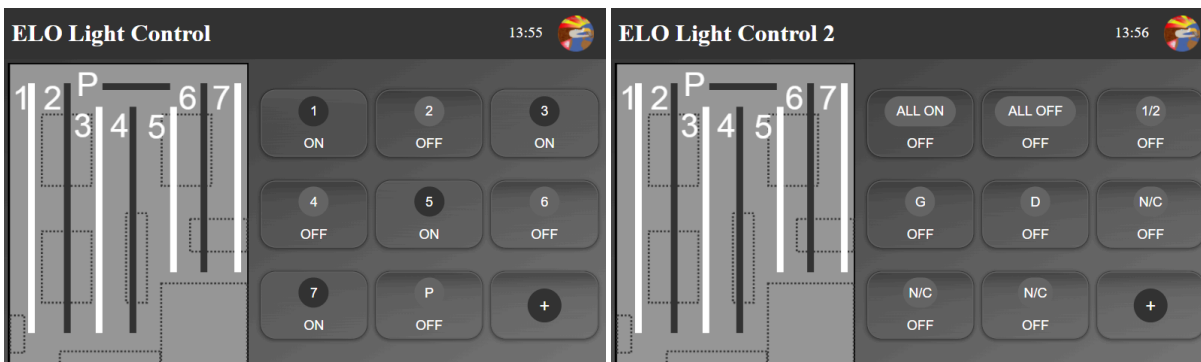
**elo** est l'utilisateur lambda, il n'a donc pas accès aux paramètres.

**admin** est un super utilisateur, il a donc un accès total.

**local** est réservé à l'écran tactile (Raspberry Pi serveur). Ce troisième utilisateur peut accéder aux contrôles des lumières sans mot de passe, mais doit l'entrer pour accéder aux paramètres de l'application.



Après s'être identifié, l'utilisateur peut modifier l'état des lumières, soit séparément sur la page 1, soit par groupes préenregistrés sur la page 2. La navigation entre les deux pages se fait via le bouton + en bas à droite.



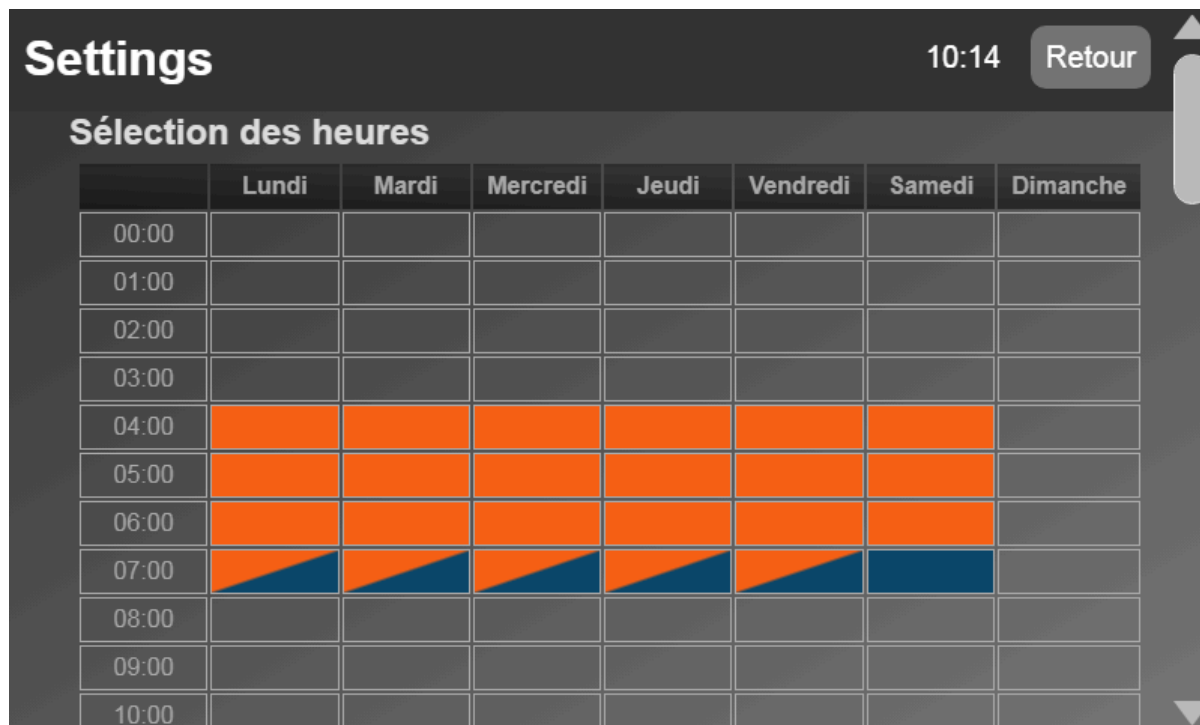
Pour accéder aux paramètres de l'application, l'utilisateur doit cliquer sur le logo ELO en haut à droite de la page.

La page **settings** contient tous les paramètres que l'utilisateur peut modifier.

En premier, on peut modifier les heures d'activation des caméras en cliquant sur la case du tableau correspondant à l'heure souhaitée. Cette case changera de couleur pour devenir orange.

Si l'on clique une deuxième fois sur cette même case, elle deviendra bleue ce qui signifie une extinction automatique des lumières et si l'on clique une troisième fois, la case deviendra orange et bleu. À ce moment, une extinction aura lieu mais les caméras resteront activées.

Si la case est grise, alors l'activation ou le déclenchement des lumières sont manuels.



Le deuxième tableau représente les lumières qui seront activées, selon le jour, si la caméra l'ordonne.

Sélection des lumières							
	Lundi	Mardi	Mercredi	Jeudi	Vendredi	Samedi	Dimanche
couloir							
Ligne 1							
Ligne 2							
Ligne 3							
Ligne 4							
Ligne 5							
Ligne 6							
Ligne 7							

Pour plus de précision, dans la plage horaire du premier tableau, on peut ajouter des heures intermédiaires à l'aide de la case **Ajouter une nouvelle heure**.

Puis la supprimer juste en dessous. Seules les heures ajoutées peuvent être enlevées.

### Ajouter une nouvelle heure

### Supprimer une heure

Le mode manuel peut être activé en cliquant sur le bouton de droite. Ce mode désactive toutes les actions non humaines sur le système. Donc, les caméras et les extinctions automatiques sont désactivées. Il est automatiquement désactivé à minuit.

### Mode manuel

Le mode manuel désactive les caméras tant qu'il est activé ou jusqu'à minuit.

On peut aussi sélectionner l'heure et la date manuellement. Cette action est aussi réinitialisée à minuit.

### Date et heure manuel

Le mode date et heure manuelle est activé jusqu'à minuit.

La section **caméras** montre les informations importantes à propos des caméras comme leur état de connexion, leur adresse IP et leur température si celle-ci devient critique.

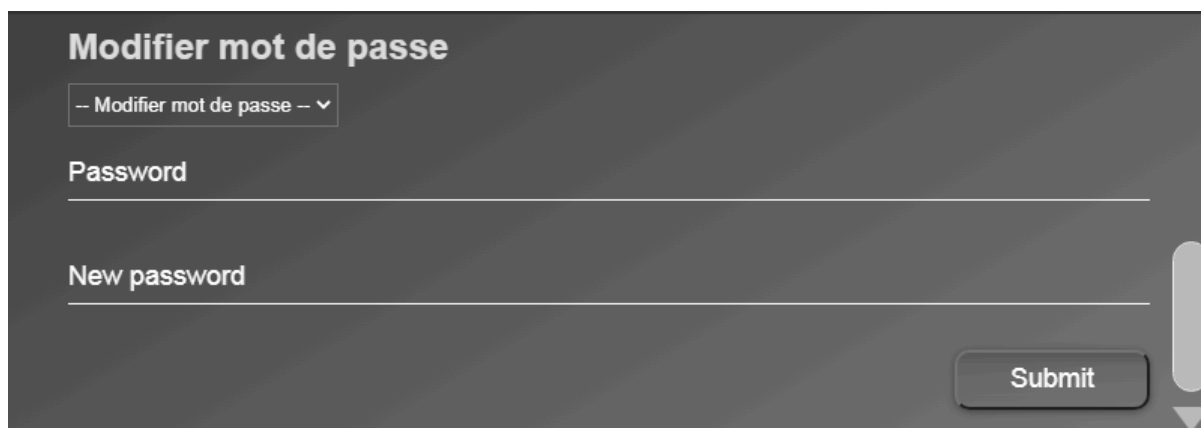


**Caméras**

Caméra 1 : **Online** IP : 172.16.32.169

Caméra 2 : **Offline** IP : 172.16.32.200

La dernière section est la section de modification des mots de passe. Pour le modifier, il faut sélectionner dans la liste déroulante un des trois utilisateurs puis entrer son mot de passe actuel et son nouveau mot de passe. Pour l'utilisateur **local**, seuls des mots de passe avec des chiffres peuvent être utilisés.



**Modifier mot de passe**

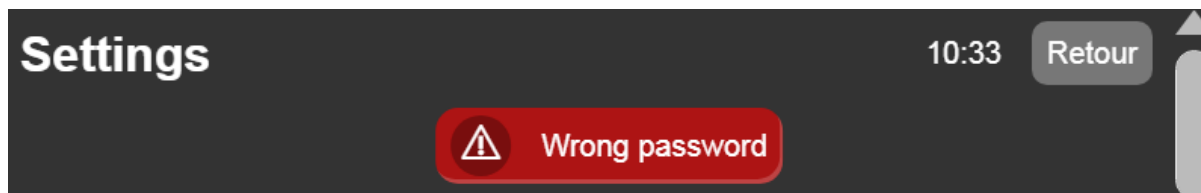
-- Modifier mot de passe -- ▾

Password

New password

Submit

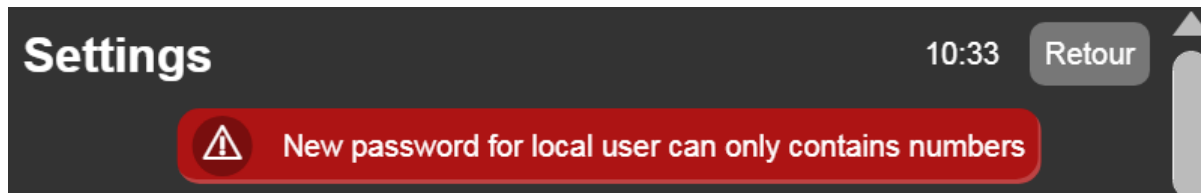
Si un mauvais mot de passe est entré, vous verrez apparaître l'erreur suivante :



**Settings** 10:33 Retour

⚠ Wrong password

Dans le cas où le nouveau mot de passe pour l'utilisateur **local** contiendrait des lettres, l'erreur suivante apparaîtra:



**Settings** 10:33 Retour

⚠ New password for local user can only contains numbers

## 3.0 - Matériel

### 3.1 - Raspberry Pi caméra

Nom	Nombre	Lien	Prix unité	Prix total
Raspberry Pi 3B+	2	<a href="https://www.pi-shop.ch/raspberry-pi-3-model-b">https://www.pi-shop.ch/raspberry-pi-3-model-b</a>	40	80
Camera night vision PIS-1138	2	<a href="https://www.distrelec.ch/fr/module-de-camera-de-vision-nocturne-pour-raspberry-pi-160-pi-supply-pis-1138/p/30163387">https://www.distrelec.ch/fr/module-de-camera-de-vision-nocturne-pour-raspberry-pi-160-pi-supply-pis-1138/p/30163387</a>	23.40	46.80
Alimentation 5V	2	<a href="https://www.pi-shop.ch/steckernetzteil-microusb-5v-25a">https://www.pi-shop.ch/steckernetzteil-microusb-5v-25a</a>	12.90	25.80
Boîtier imprimé en 3D	2		40	90
<b>Total</b>				<b>242.60</b>

### 3.2 - Serveur

Nom	Nombre	Lien	Prix unité	Prix total
Raspberry Pi 3B+	1	<a href="https://www.pi-shop.ch/raspberry-pi-3-model-b">https://www.pi-shop.ch/raspberry-pi-3-model-b</a>	40	40
Raspberry Pi 7" Touchscreen	1	<a href="https://www.pi-shop.ch/raspberry-pi-7-touch-screen-display-mit-10-finger-capacitive-touch">https://www.pi-shop.ch/raspberry-pi-7-touch-screen-display-mit-10-finger-capacitive-touch</a>	74.90	74.90
US/U 4.2	3	<a href="https://www.conrad.ch/fr/p/abb-knx-ghq6310070r0111-interface-us-u4-2-2326288.html">https://www.conrad.ch/fr/p/abb-knx-ghq6310070r0111-interface-us-u4-2-2326288.html</a>	83.80	251.40
SA/S 4.16.2.2	2	<a href="https://www.eibmarkt.com/ch/products/ABB-Schaltaktor-4fach-16A-REG-SA-S4-16-2-2.html">https://www.eibmarkt.com/ch/products/ABB-Schaltaktor-4fach-16A-REG-SA-S4-16-2-2.html</a>	169.60	339.20
TBLC 25-105	1	<a href="https://ch.farnell.com/tracopower/tbhc-25-105/netzteil-ac-dc-5v-4a/dp/2812892?ost=TBLC+25-105">https://ch.farnell.com/tracopower/tbhc-25-105/netzteil-ac-dc-5v-4a/dp/2812892?ost=TBLC+25-105</a>	52.00	52.00
Fibox ARCA 403015	1	<a href="https://www.conrad.ch/fr/p/armoire-de-commande-fibox-arca-403015-8120006-gris-400-x-300-x-150-polycarbonate-1-pc-s-1205423">https://www.conrad.ch/fr/p/armoire-de-commande-fibox-arca-403015-8120006-gris-400-x-300-x-150-polycarbonate-1-pc-s-1205423</a>	96.90	96.90
Carte Opto	1		32.20	32.20
<b>Total</b>				<b>888.60</b>

On se retrouve donc avec un coût total de : 1131.20 CHF.  
Un tel prix se justifie, car tout le matériel électrique doit être conforme aux normes suisses (module ABB et Boîtier).  
Le reste du matériel pour mon projet est basé sur les appareils que l'on utilise pour la formation, comme par exemple les Raspberry Pi 3B+.  
Il faut aussi noter que la plupart des composants m'ont été donnés par mon formateur.

## 4.0 - Installation

Pour faciliter l'installation, j'ai créé à la fin de mon TPI une image pour chacun de mes Raspberry Pi. Il vous suffira simplement de les installer sur une carte SD avec RPi imager et l'installation sera terminée.

Liens:

OMVSERVER\formation\TPI\TPI\_Forestier\_Gestion\_Lumiere\_KNX, **RPi\_camera.img**, **RPi\_server.img**.

### 4.1 - Raspberry Pi caméra

Si vous n'utilisez pas les images des Raspberry, voici les quelques étapes à suivre.  
Pour les Raspberry Pi caméra, tout ce qu'il nous faudra installer sera quelques packages python et cloner le GitLab.

Installation des packages python:

- opencv-python
- numpy
- requests
- datetime
- pickle

Commençons par Opencv qui prendra le plus de temps (environ une heure):

```
sudo apt install opencv-python
```

Après cela on va simplement vérifier si numpy est à la dernière version:

```
pip install numpy
```

On va installer pickle:

Remarque: cette installation ne sera plus nécessaire au vu de la modification présentée au point 5.2.

```
pip install pickle-mixin
```

Puis requests pour la communication:

```
pip install requests
```

Et finir par DateTime

```
pip install DateTime
```

On va ensuite cloner le GitLab pour récupérer tous les fichiers du projet:

```
git clone http://172.16.32.230/Forestier/tpi\_forestier\_gestion\_lumiere\_knx.git
```

Pour tester l'installation, vous pouvez exécuter le script python **main.py** dans le dossier **/5\_Programmation/camera/test/**.

## 4.2 - Raspberry Pi serveur

Pour le Raspberry Pi serveur, il faut suivre l'installation expliquée dans le XWiki : [Contrôle des lumières KNX](#) sous 8.0 - *Installation du Raspberry*.

Attention à bien changer les liens pour le dossier avec celui du bon GitLab.  
Faites aussi attention à l'adresse IP de votre serveur.

Votre serveur est normalement actif, mais il restera deux étapes à réaliser.  
Premièrement, sur le Raspberry Pi serveur, ouvrez le terminal et tapez la commande :

```
ping < IP_Raspberry_Camera >
```

en remplaçant < **IP\_Raspberry\_Camera** > par l'adresse IP de votre Raspberry Pi caméra.  
Si vous ne la connaissez pas, sur le RPi caméra, tapez la commande :

```
ifconfig
```

et regardez sous IPV4.

Ensuite il faut que l'on récupère l'adresse MAC donc sur le RPi serveur tapez :

```
arp -a < IP_Raspberry_Camera >
```

Dans le code **main.py** sous **5\_Programmation/serveur/**, modifier la liste **cam\_connect** en modifiant les adresses MAC de chacune de vos caméras.

Puis dans le code du Raspberry Pi caméra, modifiez l'adresse IP pour celle de votre serveur dans la fonction **send**.

La configuration est maintenant terminée et votre système est prêt à fonctionner.

## 4.3 - Serveur NTP

Il est aussi indiqué dans mon cahier des charges que la synchronisation de la date et de l'heure doit se réaliser à l'aide d'un serveur NTP.

Pour cela, nous allons modifier la configuration de nos Raspberry et en créer une nouvelle.

Premièrement, je vais installer le package ntp.

```
sudo apt-get install ntp
```

Il faut ensuite désactiver le système par défaut et lancer le serveur NTP.

```
systemctl stop systemd-timesyncd  
systemctl disable systemd-timesyncd  
/etc/init.d/ntp stop  
/etc/init.d/ntp start
```

Voilà le Raspberry Pi est connecté au serveur NTP. Mais il est connecté au serveur **debian.pool.ntp.org**.  
Comme indiqué dans mon cahier des charges, je vais le changer pour le connecter au serveur **ch.pool.ntp.org**.

```
nano /etc/ntp.conf
```

Et modifier serveur **0.debian.pool.ntp.org iburst** par **0.ch.pool.ntp.org iburst** ainsi de suite pour : 1, 2 et 3.

```
# pool.ntp.org maps to about 1000 low-stratum NTP servers. Your server will  
# pick a different set every time it starts up. Please consider joining the  
# pool: <http://www.pool.ntp.org/join.html>  
pool 0.ch.pool.ntp.org iburst  
pool 1.ch.pool.ntp.org iburst  
pool 2.ch.pool.ntp.org iburst  
pool 3.ch.pool.ntp.org iburst
```

On va ensuite redémarrer le service:

```
/etc/init.d/ntp restart
```

La configuration est terminée. Pour la tester, entrez la commande :

```
ntpq -pn
```

Vous devriez voir apparaître une réponse des serveurs avec le délai de la communication. Si cela est le cas, l'installation est terminée.

## 5.0 - Programmation

Mon TPI est un développement d'applications, ce qui dans mon cas représente uniquement de la programmation. J'ai donc systématiquement séparé mon travail en blocs pour pouvoir les tester séparément puis les rassembler pour la version finale.

### 5.1 - Caméra

Je présente ici mon premier développement que j'ai amélioré par la suite. Si vous souhaitez connaître directement les modifications effectuées passez aux points 5.2.

Comment détecter si des personnes se trouvent dans l'atelier ou non ?

Cette question représente l'entièreté de la complexité du travail de détection. Pour savoir si une personne se trouve dans l'atelier ou non, la meilleure méthode est de compter le nombre d'entrées et de sorties. Avec cela, je peux assurer que je ne laisserai jamais une personne dans le noir.

Pour cela, il va falloir détecter les portes de l'atelier. Mais ce n'est pas magique, on ne peut pas simplement demander au Raspberry Pi *trouve les portes* ! Sans intelligence artificielle, la seule méthode est de demander à l'utilisateur de les sélectionner.

#### 5.1.1 - Sélection des portes

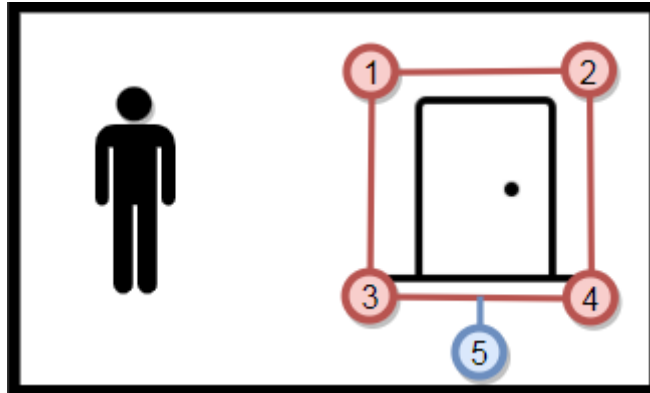
Pour détecter si une personne rentre ou sort de l'atelier, il me faut tout d'abord savoir où se trouvent les portes. Pour cela j'ai créé le code **door\_select.py** (il se trouve sous : 5\_Programmation/camera/Detection\_Porte). J'ai créé la class DoorSelect, en créant un objet DoorSelectet en lui donnant une image comme attribut, vous n'aurez plus qu'à suivre les instructions écrites dans le terminal pour réaliser la sélection des portes.

Exemple:

```
import cv2
img = cv2.imread("test.png")
d = DoorSelect(img)
```

Après avoir lancé ce code, vous verrez votre image apparaître. Il faudra, à ce moment-là, à l'aide de votre souris, cliquer (clique droit) sur les quatre coins de votre porte. L'ordre n'a pas d'importance. Puis vous devrez indiquer le sens de votre porte en ajoutant un 5 ème point pour indiquer (comme la pointe d'une flèche) la direction qu'une personne prendrait pour entrer dans la pièce.

Si vous avez réalisé cela, vous devriez obtenir quelque chose de similaire.



Si vous avez plusieurs portes, répétez l'opération de sélection.  
Une fois terminé, pressez la barre espace de votre clavier. Le système devrait vous retourner:

[INFO] Selected doors successfully saved.

À ce moment-là, vous pouvez soit presser sur la touche "enter" pour fermer le programme ou sur espace pour revenir à la sélection.  
Si le message vous indique:

[ERROR] No door select !

Alors c'est que votre sélection n'est pas correcte ou partielle. Dans ce cas, vous pouvez en cliquant sur espace, revenir à la sélection ou sur "enter" pour fermer le programme.  
Vous pouvez aussi désélectionner une porte en faisant un clique gauche avec votre souris au milieu de votre sélection.

Après avoir sélectionné la porte, le système sauvegarde les informations avec pickle. Ce fichier pourra être réutilisé pour connaître la position de la porte. Une image de la porte est aussi sauvegardée, elle a pour but de pouvoir être utilisée comme "template" de détection.

Dans la class **DoorSelect**, on retrouve 6 méthodes:

- run
- sort\_points
- draw\_door
- delete\_door
- mouse\_event

Quand on crée un objet DoorSelect et qu'on lui passe une image en argument, le programme va commencer par créer un "event handler" pour notre image, avec cela, notre fonction **mouse\_event** sera appelée à chaque fois qu'une action sera réalisée sur notre image avec la souris. Ensuite, on appelle la méthode **run** qui va agir comme un "main".

Quand l'utilisateur presse la barre espace, le programme vérifie le nombre de points qui ont été sélectionnés. Si le nombre est un multiple de 4 et que le nombre de flèches correspond, alors on va sauvegarder toutes ces informations.

```
# check if doors are selected (with an arrow)
if len(self.doors) % 4 == 0 and len(self.fleches) == len(self.doors) / 4
```

Quand on clique sur notre image, la méthode **mouse\_event** est appelée, on vérifie si "l'event" est un clique droit ou gauche puis on vient soit ajouter la position x et y de la souris à la liste **doors** ou à la liste **fleches**.



Si c'est un clique droite, on va appeler la méthode **delect\_door**. Cette méthode vient simplement regarder si les coordonnées x et y se trouvent dans une des portes, si oui on la supprime de la liste.

Pour dessiner la porte qui est en train d'être sélectionnée, la méthode **draw\_door** est là pour ça. Elle va dessiner tous les points et traits autour de la porte.

Pour s'assurer que le tour fasse un carré même si la sélection n'a pas été réalisée en carré, j'ai créé la méthode **sort\_points**. Cette méthode vient trier les points comme suite :

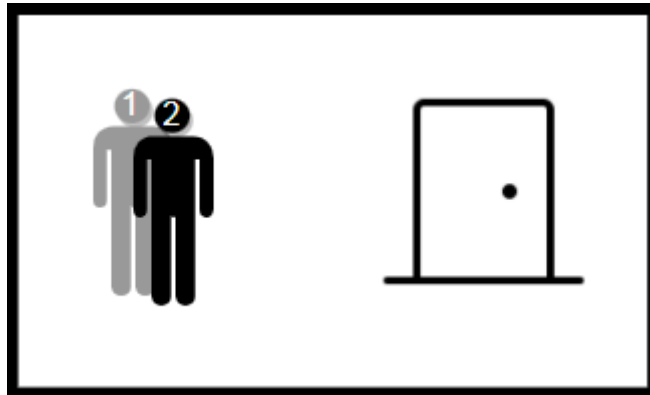
```
0    1
+ -- +
|    |
+ -- +
2    3
```

Avec tout cela la sélection des portes est terminée.

### 5.1.2 - Détection des personnes

Maintenant que nous savons où se trouvent la ou les portes, il faudra détecter les personnes en mouvement dans l'atelier.

Pour détecter un objet qui bouge dans une image, la technique la plus simple est de soustraire deux images, par ce fait seules les différences apparaîtront dans le résultat.



Pour le faire, il existe plusieurs méthodes, par exemple :  $image1 - image2$  ou `cv2.subtract(image1, image2)`. Avec ces deux premières méthodes, quand une personne se déplace, on voit apparaître sur l'image de résultat le tour de la personne et beaucoup de bruit dû à l'éclairage néon.

Pour pallier à ce problème, OpenCV ont développé des systèmes de soustraction d'arrière-plan comme: BackgroundSubtractorKNN et BackgroundSubtractorMOG2.

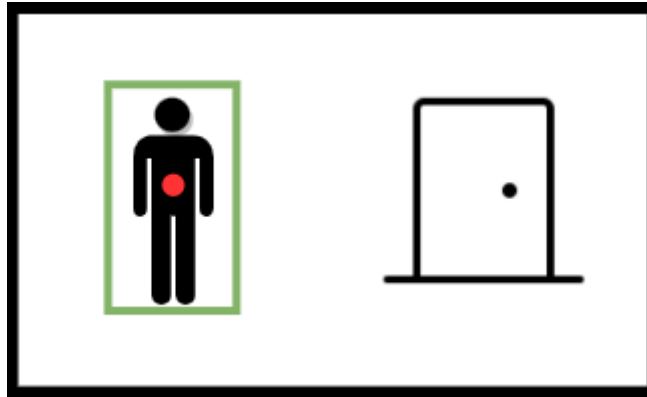
Ces deux différentes class ont pour but de soustraire l'arrière-plan mais avec plus de précisions et moins de bruit dans l'image de résultat. Pour ce faire, ils vont sauvegarder un nombre de photos et venir créer un arrière-plan mélangé de toutes les images sauvegardées, cela permet d'enlever le bruit. Elles peuvent aussi venir détecter les ombres des personnes pour ne pas les afficher comme résultat.

```
backSub = cv2.createBackgroundSubtractorKNN(history=100, dist2Threshold=500.0, detectShadows=False)
```

- history : nombre d'images sauvegardées.
- dist2Threshold : seuil sur la distance au carré entre le pixel et l'échantillon pour décider si un pixel est proche de cet échantillon.
- detectShadows : si elle est vraie, l'algorithme détectera les ombres et les marquera.

Maintenant qu'il ne nous reste que les objets en mouvement, il va falloir nettoyer notre image. Premièrement, on va réaliser un "seuillage" de notre image pour obtenir une image binaire (soit blanche soit noire). Ensuite, on va réaliser plusieurs transformations morphologiques.

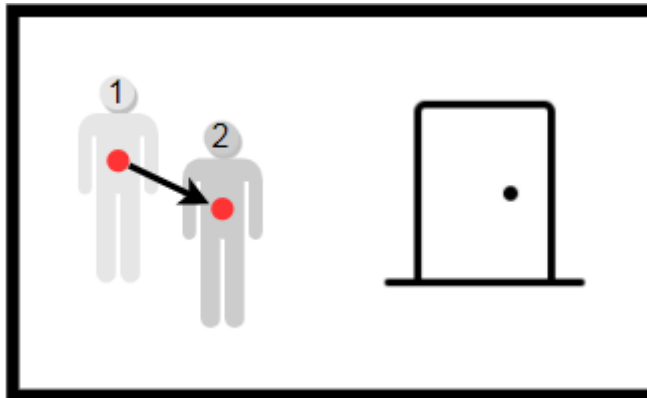
Il ne reste plus qu'à détecter les formes dans notre image avec la méthode **contour\_detect**. Dans cette méthode on va utiliser **cv2.findContours** pour détecter les contours dans l'image. Pour chacun de ces contours, on regarde si son périmètre n'est ni trop petit ni trop grand. Cette étape permet facilement d'enlever les fausses détections. On finit par dessiner le tour de notre détection avec un rectangle pour vérifier si la détection est correcte.



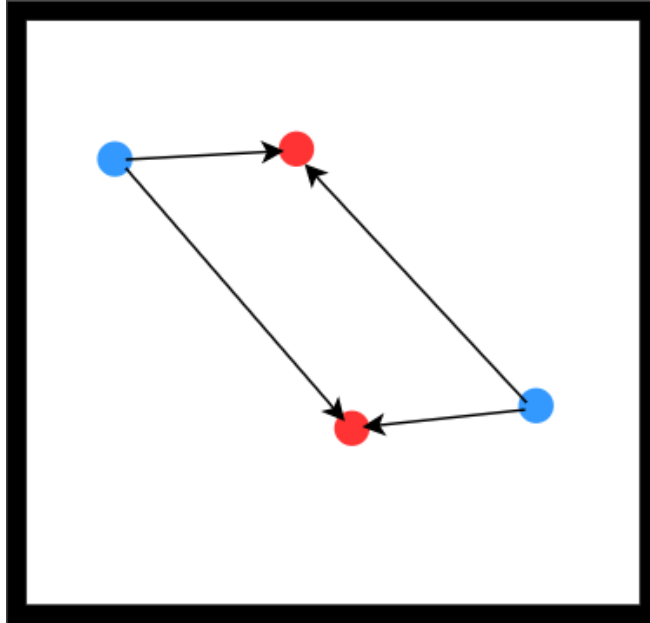
### 5.1.3 - Tracking des personnes

Maintenant que nous savons où se trouvent les personnes dans l'image, il faut réussir à connaître leur sens de déplacement.

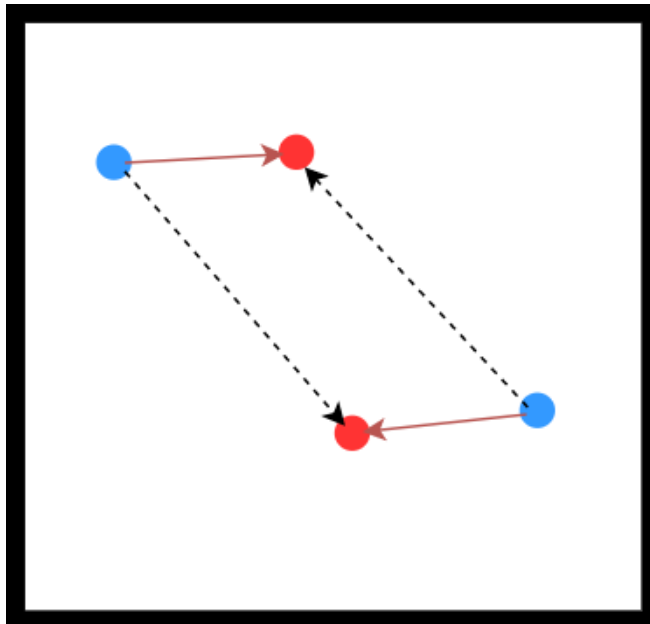
Pour cela, on va tout d'abord calculer le centre de gravité de la personne détectée. Ce point servira de représentation de la personne pour nous simplifier la vie. J'enregistre les derniers points pour pouvoir les comparer avec les derniers obtenus. Pour calculer le sens de déplacement des personnes, je cherche à trouver le point le plus rapproché.



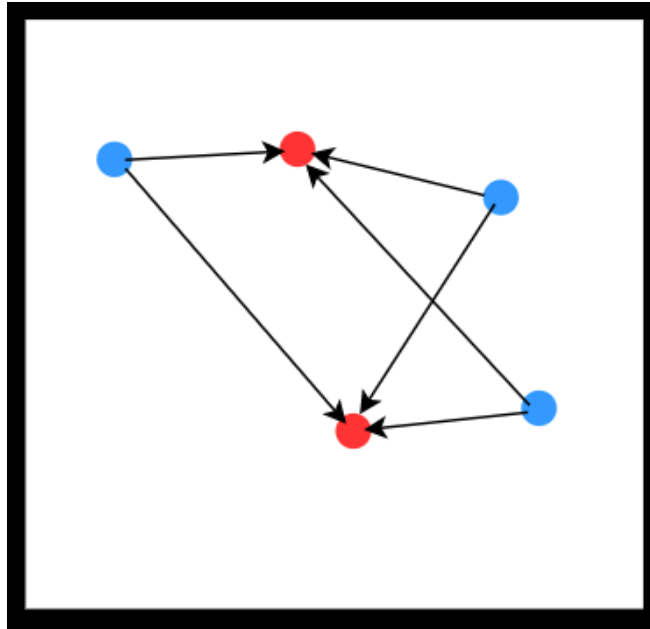
Mais dans notre atelier, il n'y a pas qu'une personne. On peut donc avoir par moment plusieurs personnes en mouvement. Dans le cas où le nombre de nouveaux points est le même que le nombre d'anciens points, cette méthode fonctionne bien.



On voit les points bleus (les anciens points) et les rouges (les nouveaux). Si l'on cherche pour chaque point bleu, le point rouge le plus proche, on obtient donc cela:



Tout fonctionne bien, rajoutons donc un nouveau point bleu.



On voit donc ici que cette méthode a quelques problèmes quand une nouvelle personne est détectée ou qu'une personne n'est plus détectée.

Pour pallier à ce problème, il faudra réaliser un tri software des informations avant de valider une entrée ou une sortie.

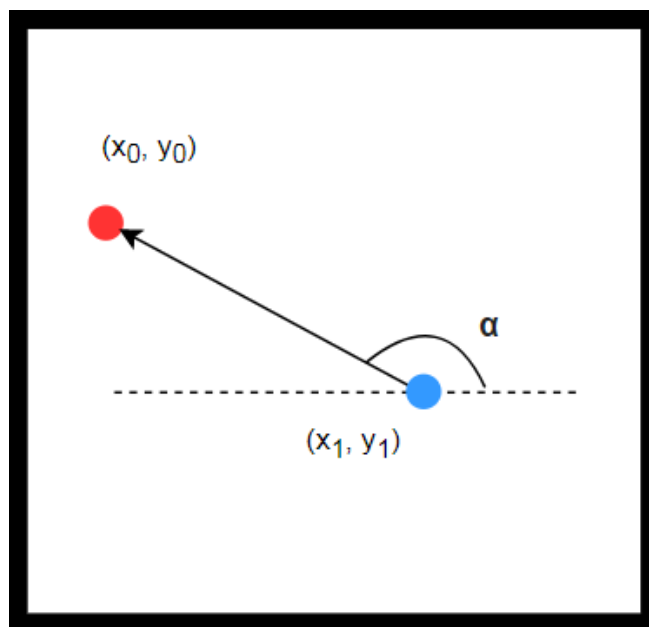
Après plusieurs jours, en travaillant sur l'amélioration, je suis revenu sur ce problème.

J'ai trouvé une solution très simple. Quand le nombre de points n'est pas le même, il suffit de trouver leur point le plus proche.

Dans le cas ci-dessus on cherche pour chaque point rouge le points bleus le plus proche.

Le point seul ne sera simplement pas relier pour le moment.

Grâce à cette flèche, on peut calculer l'angle de déplacement de la personne détectée.



Pour calculer  $\alpha$  (angle trigonométrique de la flèche), on commence par soustraire  $x_0$  par  $x_1$  et  $y_0$  par  $y_1$ , puis on utilise  $\arctan * 180 / \pi$ .

$$x = x0 - x1$$

$$y = y0 - y1$$

$$\alpha = \arctan(y / x) * 180 / \pi$$

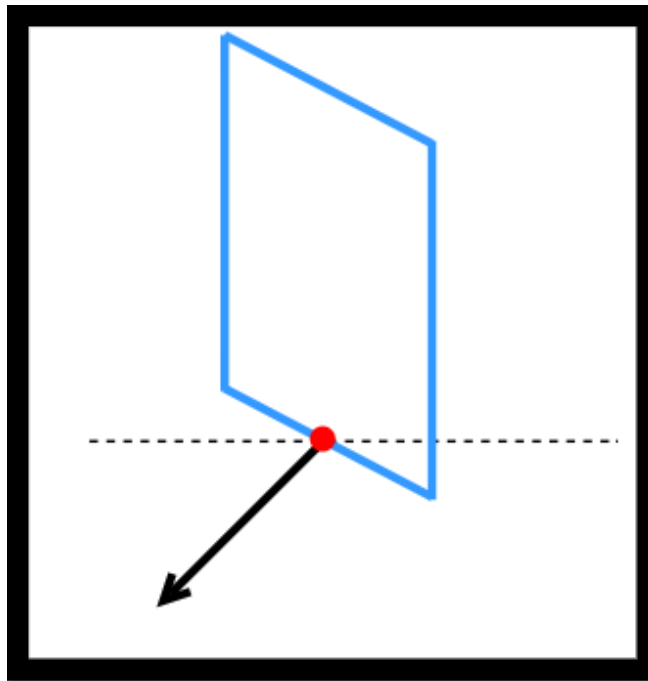
En utilisant le package math, math.atan2 nous retourne l'angle en radians, c'est pour cela qu'on ajoute  $* 180 / \pi$ . On va aussi s'assurer que l'angle ne soit pas négatif. Si c'est le cas, on lui additionne 360.

Le calcul de l'angle de déplacement marche bien si la caméra est en plongée, car si elle se trouve à la même hauteur, le déplacement à  $90^\circ$  n'est pas forcément perceptible. L'angle peut aussi par moments être très changeant d'une image à une autre, ceci est dû à la détection des mouvements.

#### 5.1.4 - Détection des entrées et des sorties

Nous avons la position des portes et le déplacement des personnes, il faut maintenant mettre ces informations en corrélation pour détecter si une personne entre ou sort.

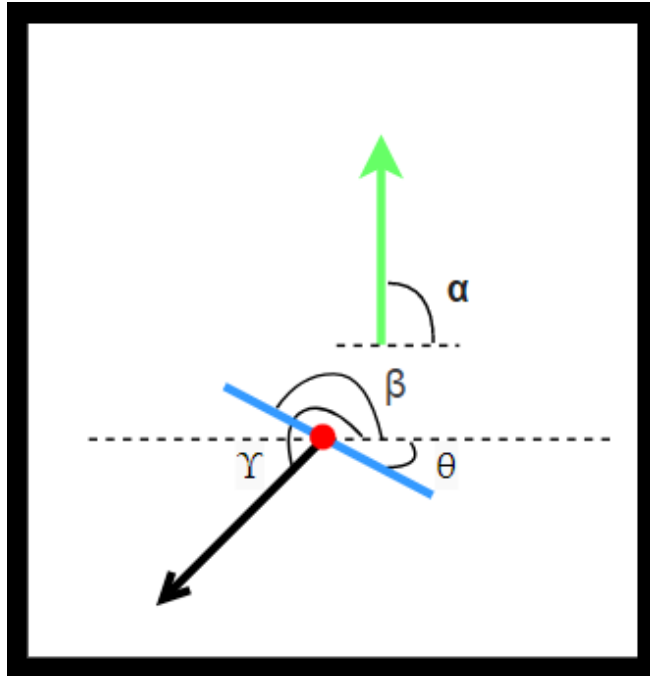
Premièrement, nous connaissons l'angle de déplacement de la personne mais pas les angles de la porte. Pour les calculer, on utilisera la sélection réalisée tout au début en lisant le fichier pickle. Les deux angles importants sont celui du bas de la porte et de la flèche indiquant dans quel sens on entre dans la pièce.



On voit sur l'image ci-dessus, le porte en bleu, le milieu du bas de la porte en rouge et la flèche en noir. Pour calculer l'angle du bas de la porte, on utilise la même formule qu'au point 5.1.3 mais nos deux points utilisés pour les calculs sont le coin en bas de la porte et le point rouge.

Passons maintenant à la partie plus complexe du problème. Il faut maintenant grâce à ces trois angles savoir si la personne entre ou sort.

Mais aussi, ajouter des angles morts pour éviter que la caméra ne détecte une personne marchant parallèlement au bas de la porte.



Quand le centre de gravité de la personne se trouvera dans la porte, on obtient une situation comme dans l'image ci-dessus.

Flèche verte : déplacement de la personne.

Trait bleu : bas de la porte.

Point rouge : milieu du bas de la porte.

Flèche noire : direction d'entrée de la porte.

On a aussi les 4 angles :  $\alpha$ ,  $\beta$ ,  $\theta$ ,  $\gamma$ .

Avant tout, on va calculer  $\theta$  qui est:

Si  $\beta - 180$  est plus grand que 0 alors  $\theta$  est égal à 360 moins  $\beta + 180$ .

Sinon,  $\theta$  vaut  $\beta - 180$ .

On va ensuite utiliser  $\theta$  pour tout mettre à plat. Cela simplifiera les calculs et évitera des problèmes d'angles négatifs.

On va ensuite vérifier si l'angle  $\alpha - \theta$  se trouve entre  $10^\circ$  et  $170^\circ$  (ces valeurs servent d'angle mort).

Si c'est le cas, on va regarder si  $\gamma - \theta$  est plus petit que  $170^\circ$ . Dans ce premier cas, la personne entre.

Si  $\gamma - \theta$  est plus grand que  $170^\circ$ , la personne sort.

Par contre, si l'angle  $\alpha - \theta$  se trouve entre  $190^\circ$  et  $350^\circ$ , on revérifie:

Si  $\gamma - \theta$  est plus petit que  $170^\circ$ . Dans ce premier cas, la personne sort.

Sinon  $\gamma - \theta$  est plus grand que  $170^\circ$ , la personne entre.

Voilà, après toutes ces maths, on sait enfin si la personne entre ou sort par la porte. Mais si on teste le code comme ceci, on obtiendra :

```
sort
sort
sort
...
```

On ne peut donc pas savoir combien de personnes sortent ou entrent.

Pour cela, j'ai créé la class **info**.

Dans cette class, on va utiliser la fonction "verify". On va créer une liste de toutes les détections avec comme informations: l'heure de la détection, si c'est une entrée ou une sortie, le numéro de la porte et le numéro de la personne.

Cette fonction va prendre en compte chaque détection et regarder celles qui l'ont précédée.

Si on détecte 3 (ou plus) détections similaires en moins de 5 secondes, le système valide cette détection.

On pourra donc l'envoyer au système pour modifier les lumières.

### 5.1.5 - Mesures

Nom	Valeurs & Utilisation
CPU	41 %
RAM	120 mb => 15 %
Temperature	80 °C
FPS	3.2

Mise à part la température du Raspberry Pi toutes ces mesures me paraissent viables.

La température est critique pour le Raspberry Pi, on atteint très rapidement ses limites. Je pense qu'un ventilateur devra être rajouté pour éviter tous problèmes.

La température de la caméra est à surveiller car de nuit avec les led infrarouges allumées, je pense que sa température va vite monter.

### 5.1.6 - Tests et améliorations

J'ai donc réalisé une multitude de tests pour vérifier le fonctionnement de mon système.

Après avoir sélectionné les trois portes de l'atelier, j'ai fait plusieurs allers-retours à travers les portes.

Pour la porte bleue (menant à la salle de théorie), mes résultats sont excellents. La détection est correcte et sans erreur.

Pour la porte rouge, mes résultats sont complètement faux. Quand on ouvre la porte, le changement de lumière crée une grande différence et la caméra ne voit plus les personnes.

Pour la 3ème porte, la détection est fonctionnelle mais pas optimale, ceci est dû à la distance et au fait que la caméra a une vision moins en plongée que pour la porte bleue.

Pour obtenir une détection optimale voire parfaite, je pense que placer les caméras directement au-dessus de la porte regardant le sol simplifierait la détection en enlevant la sélection des portes et augmenterait la précision de détection.

Cette méthode permettrait aussi l'utilisation de la vision nocturne, car la distance serait plus petite (moins de 3 mètres).

## 5.2 - Amélioration réalisée

Comme expliqué au point 5.1.6, je souhaitais placer les caméras au-dessus des portes et modifier le système de détection pour le rendre plus précis.

Au vu de mon planning, j'ai eu le temps de réaliser cette amélioration.

Le fonctionnement du système reste en grande partie le même. La plus grande modification est que je n'ai plus à détecter les portes, ni à calculer les angles. Cela va, en grande partie, simplifier mon code et la compréhension du système pour toutes autres personnes si intéressant.

### 5.2.1 - Détection des personnes

Pour la détection des personnes, je garde le même code qu'auparavant avec quelques petites modifications.

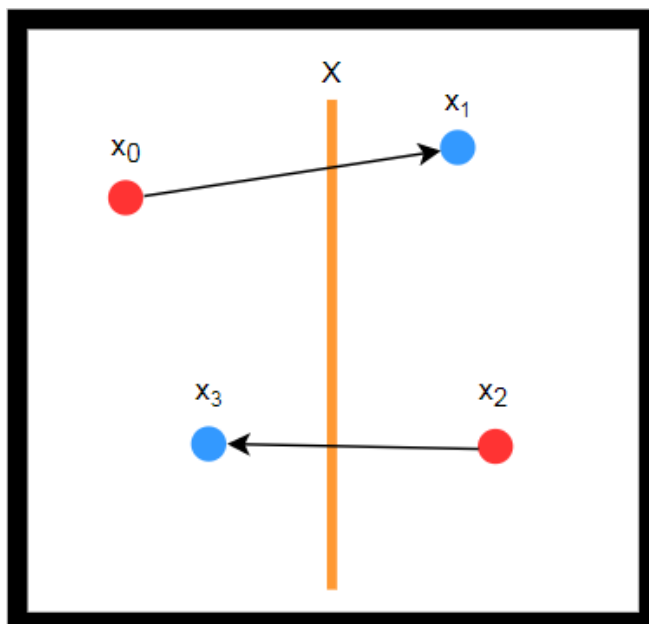
Premièrement, dans la fonction `contour_detect`, j'ai augmenté la comparaison pour la taille du contour minimum. En sachant que la caméra est plus près des personnes, celles-ci sont donc plus grosses à l'image. En augmentant la valeur minimum, je supprime une grande partie des parasites.

Deuxièmement, dans `personne_detect`, j'ai ajouté une quatrième transformation morphologique. En ajoutant une "érosion" pour supprimer les petites impuretés et j'ai modifié quelques paramètres pour ajuster la détection.

### 5.2.2 - Tracking des personnes

Pour le tracking des personnes, j'ai complètement supprimé le calcul des angles et je l'ai remplacé par une simple comparaison.

Cette comparaison vérifie si une personne a franchi le milieu de l'image, si c'est le cas, alors on peut savoir si elle entre ou sort.



Sur l'image ci-dessus on voit en rouge l'ancien point et en bleu le nouveau point. On va simplement comparer leur coordonnée en X.

### 5.2.3 - Détection des entrées et des sorties

Je vais uniquement garder la fonction **send** du fichier `entry_detect`, et simplement venir récupérer la liste *inout* qui se trouve dans l'objet *tracking*. Je viens simplement parcourir cette liste et j'envoie la valeur lue au serveur. Finalement je vide la liste.

### 5.2.4 - Tests et améliorations

Après plusieurs tests, j'ai conclu que cette amélioration était utile au système car elle apporte plusieurs corrections de bug. Elle permet d'utiliser la vision nocturne des caméras. Elle améliore la détection et elle permet un comptage des personnes.

Pour voir un exemple vidéo:

**OMVSERVER\formation\TPI\TPI\_Forestier\_Gestion\_Lumiere\_KNX\Exemple\exemple\_1.mp4**

Et un exemple de nuit:

**OMVSERVER\formation\TPI\TPI\_Forestier\_Gestion\_Lumiere\_KNX\Exemple\exemple\_2.mp4**

### 5.2.5 - Bugs

Un bug est survenu après un test durant une nuit complète. Le système se stoppait sans aucune erreur.

J'ai donc testé plusieurs choses pour pallier à ce bug.

Premièrement, j'ai affiché la température de mon Raspberry pour vérifier si elle n'excédait pas la valeur limite.

Deuxièmement, j'ai testé d'exécuter le programme en arrière-plan pour qu'il ne puisse pas être interrompu par l'utilisateur.

Aucun changement visible.

J'ai aussi testé d'autres programmes mais sans résultats concluants. Pour finir, j'ai placé un ventilateur devant mon Raspberry Pi et la caméra.

Le programme a pu fonctionner plus longtemps, mais s'est quand même arrêté.

Je pense que le problème vient de la température de la caméra.



J'ai réalisé plusieurs mesures de la température sur la caméra, j'ai pu constater une forte augmentation de la température quand les LED sont allumées. Jusqu'à 70 °C après une heure de fonctionnement. Après avoir ajouté des dissipateurs thermiques à l'arrière de la caméra, un derrière chaque LED et deux petits derrières la caméra, la température ne monte pas au-dessus de 50 °C. Cette température est beaucoup plus raisonnable. L'ajout de ventilateur peut être une solution à envisager à long terme.

## 5.3 - Serveur

Pour la base du serveur, j'ai utilisé la version 1.2 de mon projet Controle des lumières KNX.

[GitLab V1.2](#)

Cette version est une version stable et testée.

Toute mon interface est basée sur un serveur web. Grâce à ce serveur, les utilisateurs peuvent s'y connecter facilement via un navigateur s'ils sont connectés au Wi-Fi.

Le serveur sera aussi utile pour la communication avec les caméras.

Le serveur est développé en Python avec le module Flask (<https://flask.palletsprojects.com/en/2.0.x/>). Flask peut uniquement être utilisé comme serveur de développement, c'est pour cela que j'utilise GUNICORN (<https://gunicorn.org>), qui lui est un vrai serveur WSGI (Web Serveur Gateway Interface).

J'ai aussi paramétré le proxy NGINX (<https://www.nginx.com>), pour pouvoir sécuriser mon site en le passant en HTTPS.

Tout cela est expliqué dans ma documentation :

(<https://xwiki.serverelo.org/xwiki/bin/view/Centre%20de%20Formation%20ELO/Projets/Controle%20des%20lumières%20KNX/>)

### 5.3.1 - Fonctions déjà implémentées (V 1.2)

- Login avec mot de passe.
- Gestion des Sessions.
- Gestion des erreurs HTTP.
- Proxy HTTPS.
- Contrôle des GPIO du Raspberry Pi.

### 5.3.2 - Fonctions à implémenter pour le TPI

- Page de login pour l'écran tactile.
- Gestion des plages horaires de la caméra.
- Modification du mot de passe.
- Mode manuel.
- Horaire manuel.
- Informations des caméras.
- Extinction automatique.

### 5.3.3 - Page settings

La page settings est une page réservée aux administrateurs et à l'écran tactile.

Pour y accéder, il suffit de cliquer sur le logo de notre atelier en haut à droite de l'écran.

Si vous êtes logué en utilisateur **elo**, vous serez redirigé vers la page de login.

Par contre, si vous souhaitez y accéder via l'écran tactile, une page (login2) vous demandera d'entrer un mot de passe.

### 5.3.4 - Page login 2

Cette page est uniquement accessible si un utilisateur souhaite accéder au setting via l'écran tactile.

Pour qu'un utilisateur puisse rentrer un mot de passe sans avoir de clavier, j'ai dû créer un clavier en HTML et CSS.

J'ai donc choisi que le mot de passe serait uniquement composé de chiffres par manque de place sur l'écran tactile.

La page est donc constituée d'un en-tête avec le titre, des alertes (s'il y en a), de l'heure et d'un bouton retour qui nous ramène à la page 1.

Au centre, sur le clavier pour entrer le mot de passe, on y retrouve les 10 chiffres (0 à 9), un bouton **Effacer** et un **Submit**.

Pour l'HTML, on a simplement 12 boutons avec chacun une valeur et un texte:  
Voici le bouton "1":

```
<button type="submit" class="btn" name="btn" value="1"> <span class="btn_num">1</span> </button>
```

Pour afficher le clavier, en CSS je viens créer une grille:

```
display: grid;
grid-template-columns: repeat(3, 1fr);
```

Du côté python, on retrouve un code très simple.  
Premièrement on récupère la valeur du bouton.

```
button_click = request.form.get('btn')
```

À chaque nouvelle touche pressée, on vient ajouter le numéro dans une liste.

```
mot_de_pass.append(button_click)
```

Si le bouton 11 (Submit) est pressé on va joindre la liste et la comparer avec le password actuel de l'utilisateur **local**.

```
mot_de_pass = "".join(mot_de_pass)
user = [x for x in users if x.username == "local"][0]
```

Si le mot de passe est correcte, on va rediriger l'utilisateur sur la page en transmettant la variable authorized à 1.

```
return redirect(url_for('settings', authorized=1))
```

Si le bouton 10 (Effacer) est pressé on va simplement vider la liste.

```
mot_de_pass = []
```

### 5.3.5 - Plage horaire

Il m'est demandé dans mon cahier des charges de gérer:

- Les plages horaires où les caméras peuvent commander les lumières.
- Quelles lumières seront enclenchées.
- À quel moment une extinction automatique se produira.

Pour la gestion des plages horaires, j'ai choisi de suivre l'exemple qui m'a été donné dans mon cahier des charges.

J'ai donc réalisé deux tableaux avec comme entête les 7 jours de la semaine.

Le premier tableau permet de sélectionner les heures d'activation des caméras et les extinctions automatiques durant chaque jour.

Le deuxième tableau indique quelles lumières allumer si la caméra le demande.

La création de ces deux tableaux se fait dans le fichier **table.py** sous /5\_Programmation/server/table/.

À l'intérieur du fichier on trouve deux class, la première **Table** et la seconde **Table2** qui hérite de **Table**.

Ces deux class gèrent l'enregistrement, la modification et l'envoi des informations au fichier main. Pour la sauvegarde des tableaux, j'utilise pickle, en enregistrant dans le sous-dossier **settings** cinq fichiers pour enregistrer toutes les valeurs des différentes tables.

Si l'on souhaitait réinitialiser les deux tableaux, il suffirait simplement de supprimer les fichiers. Après le redémarrage, ils seront recréés mais sans aucune valeur.

Pour la création du tableau en HTML, j'utilise JINJA (<https://jinja.palletsprojects.com/en/3.0.x/>). JINJA permet de récupérer les variables envoyés depuis le fichier python et de les afficher. On peut aussi venir écrire du code directement dans notre fichier HTML, comme des "IF" ou des boucles "FOR".

Pour afficher mes tableaux, je viens réaliser une boucle "for" pour chaque ligne.

```

1 <!--Table 1-->
2 <form method="post">
3 <!--Title-->
4 <h2>Sélection des heures </h2>
5 <table class="table_1">
6 <!-- header -->
7 <tr>
8 <th id="hour"></th>
9 <th>Lundi</th>
10 <th>Mardi</th>
11 <th>Mercredi</th>
12 <th>Jeudi</th>
13 <th>Vendredi</th>
14 <th>Samedi</th>
15 <th>Dimanche</th>
16 </tr>
17
18 {% for tab in table %}
19 {% set iteration = loop.index0 %}
20 <tr>
21 <!-- hour -->
22 <td>{{ heure[iteration] }}</td>
23 <!-- row of btn -->
24 {% for t in tab %}
25 <td id="n{{ t }}">
26 <button type="submit" name="btn_tbl_1" value="{{ iteration }}.{{ loop.index0 }}">
27 &#8203;
28 </button>
29 </td>
30 {% endfor %}
31 </tr>
32 {% endfor %}
33 </table>
34 </form>

```

Ci-dessus le code permettant d'afficher le tableau numéro 1. Premièrement, on voit que le tout se trouve dans un `form`, c'est grâce à lui que l'on pourra envoyer les informations. Ensuite on voit le titre `h2`. Après le titre, on a le tableau (`table_1`). On vient ensuite afficher les jours de la semaine, en laissant vide la première case pour ne pas afficher de jour au-dessus de la colonne des heures.

Puis on voit `{% for tab in table %}`, ce code va donc parcourir la variable `table`. Cette variable a été envoyée grâce au code :

```
return render_template('settings.html', table=table)
```

Grâce à cette technique, je n'ai donc pas besoin d'écrire chaque ligne du tableau et je peux facilement en ajouter ou en enlever.

Pour l'affichage des heures sur la gauche du tableau, j'utilise la même méthode mais cette fois en utilisant l'index de la boucle "for".

Puis je viens créer une deuxième boucle "for" qui affiche chaque case avec un id valant 0, 1, 2 ou 3 selon sa couleur (0 gris, 1 orange, ...).

Dans cette case, on place un bouton avec une valeur (value) écrite grâce aux boucles "for". Exemple 0.0 puis 0.1 étant les deux premières cases en haut à gauche, le premier nombre étant la ligne et le deuxième la colonne.

Dans chaque bouton, on écrit `&#8203;` ; étant simplement un espace pour assurer que le CSS l'affiche correctement.

Voilà un exemple concret avec la première ligne du tableau.

Rendu sur la page:

Sélection des heures							
	Lundi	Mardi	Mercredi	Jeudi	Vendredi	Samedi	Dimanche
00:00							

HTML de la page:

```

1 <h2>Sélection des heures </h2>
2   <table class="table_1">
3     <tr>
4       <th id="hour"></th>
5       <th>Lundi</th>
6       <th>Mardi</th>
7       <th>Mercredi</th>
8       <th>Jeudi</th>
9       <th>Vendredi</th>
10      <th>Samedi</th>
11      <th>Dimanche</th>
12    </tr>
13
14    <tr>
15      <td>00:00</td>
16      <td id="n0"> <button type="submit" name="btn_tbl_1" value="0.0"> &#8203; </button> </td>
17      <td id="n1"> <button type="submit" name="btn_tbl_1" value="0.1"> &#8203; </button> </td>
18      <td id="n2"> <button type="submit" name="btn_tbl_1" value="0.2"> &#8203; </button> </td>
19      <td id="n3"> <button type="submit" name="btn_tbl_1" value="0.3"> &#8203; </button> </td>
20      <td id="n2"> <button type="submit" name="btn_tbl_1" value="0.4"> &#8203; </button> </td>
21      <td id="n1"> <button type="submit" name="btn_tbl_1" value="0.5"> &#8203; </button> </td>
22      <td id="n0"> <button type="submit" name="btn_tbl_1" value="0.6"> &#8203; </button> </td>
23    </tr>

```

Pour le deuxième tableau, j'utilise exactement la même technique. En modifiant évidemment le nom de mes variables affichées.

### 5.3.6 - Ajout d'une heure

Comme indiqué au point 3.2 - *Serveur*, si on souhaite plus de précision dans le premier tableau, on peut ajouter des heures intermédiaires.

Ces heures seront automatiquement ajoutées dans le tableau mais seront aussi sauvegardées à part.

Grâce à cela, quand on souhaite supprimer une heure, dans la liste déroulante, il apparaît seulement les nouvelles heures ajoutées mais pas les heures fixes.

L'ajout et la suppression d'heures se fait dans le fichier **table.py** et sont stockés dans un fichier pickle.

t.add\_custom\_line(nouvelle\_heure)

Du côté de l'HTML, on retrouve simplement un input de type time.

```
<input type="time" id="time" name="time" required>
```

Pour la suppression des heures, j'utilise de nouveau une boucle "for" pour afficher chaque heure dans la liste déroulante.

On peut facilement voir l'utilité de la boucle for pour l'affichage des tableaux, ce qui les rend parfaitement modifiables sans toucher au code HTML.

### 5.3.7 - Mode manuel

Le mode manuel vient désactiver toutes les actions "automatiques", comme les actions des caméras ou les extinctions automatiques.

Quand un utilisateur active le mode manuel, on va simplement changer la valeur de **mode\_manuel** à 1. On utilise simplement cette variable quand la caméra nous envoie des requests ou quand le programme vérifie les extinctions automatiques.

Pour l'affichage sur la page WEB, on utilise le CSS pour modifier son affichage.

```
<button type="submit" name="manuel" class="c{{ mode_manuel }}" value="{{ mode_manuel }}"></button>
```

Si sa class est **c0** alors il est désactivé si c'est **c1** alors il est activé. On va aussi modifier sa valeur.



*Éteint à gauche, allumé à droite.*

Pour inverser sa valeur, j'utilise ce simple code:

```
mode_manuel = 1 - int(manuel)
```

À minuit, le mode manuel se désactive et repasse à 0 (désactivé).

### 5.3.8 - Heure manuelle

Pour la gestion de l'heure et de la date en mode manuel, j'utilise le package python DateTime (<https://docs.python.org/3/library/datetime.html>). Quand un utilisateur vient à modifier l'heure et la date, je récupère le jour qu'il a choisi sous forme d'un nombre, 0 étant lundi et 6 étant dimanche et je vais récupérer la nouvelle heure. Avec l'heure entrée par l'utilisateur, je calcule la différence de temps avec l'heure réelle. Chaque fois que mon programme demandera l'heure et que le mode heure manuel sera activé, il recevra l'heure réelle additionnée avec la différence de temps calculée précédemment.

```
x = datetime.fromisoformat(heure_man)
diff = x - datetime.strptime(datetime.now().strftime("%H:%M"), "%H:%M")
diff = timedelta(seconds=diff.seconds)
```

Et pour le jour:

```
mode_heure_manuel[1] = x.weekday()
```

Pour la partie HTML, on retrouve un input "datetime-local".

```
<input type="datetime-local" id="heure_manuel" name="heure_manuel" {{ var }}>
```

Var est une variable créée deux lignes en dessus. C'est simplement une chaîne de caractères contenant "required" si le mode heure manuel est désactivé. Sinon elle ne contient rien.

Cela oblige l'utilisateur à sélectionner une date s'il veut activer le mode mais pas s'il souhaite le désactiver.

Pour le bouton à droite, on utilise la même méthode que le bouton mode manuel.

À minuit, le système va automatiquement réinitialiser toutes les valeurs et repasser sur l'heure réelle.

### 5.3.9 - Tâche en arrière plan

La fonction de tâche en arrière-plan avait déjà été implémentée dans la version avant le TPI. Mais je l'ai modifiée en lui ajoutant plusieurs utilités.

La fonction **run\_job** est lancée en parallèle du serveur grâce à un *thread*.

Cette fonction s'exécute toutes les minutes, elle est utilisée pour les extinctions automatiques, qu'elles soient demandées par les caméras ou par le tableau des heures, elle sert aussi pour la réinitialisation à minuit.

J'utilisais **time.sleep(60)** pour attendre une minute entre chaque exécution de la fonction **run\_job** mais avec l'ajout du ping des caméras et l'imprécision du timer, j'ai préféré modifier cela et vérifier dans une boucle while si l'on est passé à une nouvelle minute. Cette modification fait que les actions d'extinction se réaliseront directement aux changements de minute et pas aléatoirement.

Pendant la réinitialisation à minuit, je m'assure que trois variables soient réinitialisées:

- Le nombre de personnes dans l'atelier.
- Le mode manuel.
- L'heure manuelle.

Ensuite, pour réaliser les extinctions automatiques, le programme vérifie si le mode manuel est déclenché, puis il va en premier regarder si la case du tableau correspond à l'heure actuelle, si c'est le cas, il regarde la couleur. Si la case est bleue alors il éteint toutes les lumières de l'atelier.

Pour vérifier si la caméra a demandé une extinction automatique, le programme va lire la liste **cam\_send\_turn\_off** avec une boucle for. Il regarde si l'heure enregistrée dans la liste additionnée avec **TURNOFFTIME** (constante de temps avant d'éteindre les lumières) est égale à l'heure actuelle. Si c'est le cas, il parcourt le tableau des heures dans le sens inverse pour récupérer la case du tableau dans laquelle on se trouve.

Puis il regarde si sa couleur est orange, si c'est le cas alors il éteint les lumières.

## 5.4- Communication Wifi

Pour que les Raspberry Pi caméra puissent envoyer des informations au serveur, j'ai utilisé le serveur flask déjà en fonction comme receveur.

Le raspberry pi caméra vient envoyer une request "post" au serveur sur une page cachée. Cela permet une communication fiable avec une gestion des erreurs et un envoi automatique de feed back.

Pour le faire, j'utilise le package "requests" et j'envoie la trame suivante au serveur:

```
$,RPWCSD,019,10:100351,0*
```

Cette trame correspond au standard de l'atelier et respecte le [protocole de communication ELO](#).

Le \$ et \* servent de début et de fin de trame.

les , servent de séparateur.

**RPWC xx** signifie: Raspberry Pi, Wifi, Caméra.

xx peut être remplacé par :

**SD** signifie send data.

**OK** signifie ok pas de problème.

**ER** signifie error.

La troisième partie indique la longueur des données.

La quatrième partie contient les données envoyées: **10:100351**.

Dans mon exemple, les données envoyées sont:

- l'heure: 10:10
- La température sur trois nombres: 035 (35°C)
- Et l'action à effectuer: Allumer: 1, Eteindre: 0.

Le 0 à la fin peut être utilisé comme sécurité. Mais je ne l'utilise pas pour ce projet.

Pour que les caméras puissent envoyer leurs informations, j'ai créé une page /camera. Pour que cette page soit uniquement accessible par les caméras, j'ai d'abord pensé utiliser leur adresse IP, mais après discussion avec Monsieur Dupertuis, il m'a conseillé d'utiliser les adresses **MAC** des Raspberry Pi pour éviter tous problèmes dans le cas d'une modification d'adresse IP des Raspberry Pi.

Pour connaître l'adresse MAC des Raspberry Pi, il faut entrer la commande suivante dans le terminal:

```
arp -a
```

Vous devriez voir afficher des adresses IP et des adresses MAC juste à côté.

Dans mon code python, j'ai créé une liste **cam\_connect** qui contient pour chaque caméra, si elle est connectée, son adresse IP, son adresse MAC et sa température.

Dans mon code python, le programme vérifie si l'adresse MAC existe avec la fonction **getmacadd** puis il vérifie dans la liste si elle existe.

```
if not any(mac_add in x for x in cam_connect) or mac_add is False:  
    abort(404)
```

Après ces vérifications, il lit les données envoyées par la caméra pour vérifier si elles correspondent. Il modifie le nombre de personnes dans l'atelier et allume les lumières s'il le faut en vérifiant dans le tableau des settings.

Pour tester la communication, un code test\_communication.py dans /5\_Programmation/camera/test/ est là pour ça.

J'ai pu remarquer un problème pendant les tests, le serveur n'était pas capable d'exécuter les commandes système.

Pour régler ce problème, j'ai dû modifier la configuration du serveur GUNICORN, voire 8.1.4 - Configuration Gunicorn dans mon ancien projet.

<https://xwiki.serverelo.org/xwiki/bin/view/Centre%20de%20Formation%20ELO/Projets/Controle%20des%20lumières%20KNX/#H8.1.4-ConfigurationGunicorn>



## 6.0 - Remerciements

Je tiens à remercier toutes les personnes qui m'ont apporté leur aide lors de mon apprentissage, de la réalisation de ce projet ainsi qu'à l'écriture de ce TPI.

Je voudrais tout d'abord remercier mes formateurs M. Dupertuis et M. Perrin, pour le suivi durant mes 4 années d'apprentissage, leur patience et leurs judicieux conseils.

Les experts M. Perrelet et M. Wuermli pour leur accompagnement tout au long du projet.

Je désire aussi remercier mes collègues pour le bon climat de travail dans l'atelier, ainsi que leur aide et soutien durant mon apprentissage.

Enfin, j'adresse un grand merci à ma maman pour la relecture de ce dossier.

## 7.0 - Conclusion

Pour terminer, je suis fier d'avoir réalisé ce projet. Il s'agissait de mon premier projet concret. Dans l'ensemble, mon TPI c'est très bien déroulé. J'ai cependant dû faire une modification conséquente vers la fin de mon travail, ce qui m'a fait réaliser que ma première idée n'était pas parfaite et qu'au moment de tester mon système tout ne fonctionnait pas.

Grâce à ce projet, j'ai pu mettre en pratique les connaissances que j'ai acquises en programmation au cours de ces dernières années. J'espère que ce projet durera dans le temps et pourra peut-être être amélioré par de futurs apprenants.

## 8.0 - Bugs connus et améliorations

### Bugs:

- Le programme des caméras cesse de fonctionner après une longue période de temps.

### Améliorations:

- J'aimerais modifier la partie détection des entrées / sorties en plaçant les caméras directement en-dessus des portes.