Chapitre 23 - La commande du réseau

Nous allons maintenant voir comment gérer des sorties de manière à pouvoir commander le réseau.

Deux fonctions sont prévues pour envoyer un signal sur une sortie :

digitalWrite qui permet de mettre la sortie numérique à l'état bas ou à l'état haut

analogWrite qui permet d'envoyer un signal PWM de rapport cyclique réglable de 0 à 100% (en théorie car le microcontrôleur ne permet pas d'atteindre vraiment ces deux bornes mais s'en rapproche beaucoup). Le signal reste présent sur la sortie si on ne fait plus rien et n'a pas besoin d'être entretenu.

Le courant délivré par les sorties est généralement trop faible pour commander directement le réseau sauf s'il s'agit d'allumer des LED (et encore, l'ensemble du module est limité à 200 mA au total sur toutes ses broches). Il doit donc être amplifié. Le chapitre 21 nous a montré comment utiliser un transistor ou un circuit intégré ULN2803 pour amplifier le signal afin d'actionner un ou plusieurs relais qui renvoient le courant traction dans la voie aux bons endroits ou bien qui actionnent des animations sur le réseau.

Le signal peut aussi être amplifié pour faire tourner un moteur. Dans ce cas, sa vitesse de rotation peut être réglée par le rapport cyclique du signal envoyé à l'entrée de l'ULN2803.

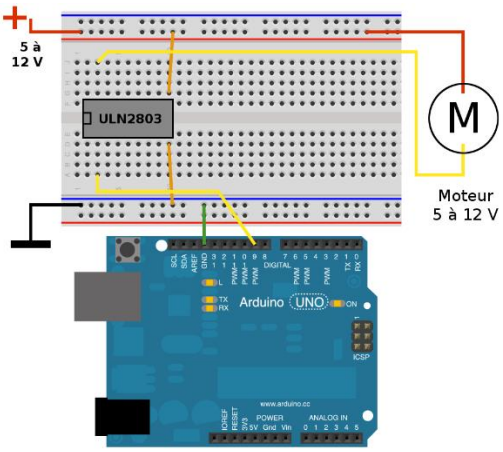

Figure 23. 1

Le montage de la figure 23.1 permet de commander un moteur mais dans un seul sens de rotation ; par exemple, un manège de fête foraine qui commence à tourner lentement puis s'accélère avant de revenir à sa vitesse d'arrêt et de faire une pause pour que les gens descendent.

Toutes les animations du réseau qui requièrent un mouvement mécanique de rotation, peuvent être aisément commandées par des moteurs pas à pas puisqu'il existe la bibliothèque Stepper qui permet de commander ce genre de moteurs. Le site Arduino donne des schémas de montage à base de moteurs pas à pas ainsi que les programmes nécessaires. Il y a deux types de moteurs pas à pas : les moteurs unipolaires et les moteurs bipolaires. Les deux types peuvent être utilisés avec Arduino même si l'interface électronique n'est pas tout à fait la même. Référez-vous au site Arduino pour de plus amples informations. Les moteurs pas à pas bipolaires peuvent être récupérés dans le matériel informatique réformé : lecteur de disque, imprimante, scanner.

Les servos utilisés en modélisme traditionnel, permettent de commander des mouvements ; grâce à la bibliothèque Servo, leur utilisation est extrêmement simple et on peut arriver à des positionnements

extrêmement précis. On en trouve de bon marché et on peut les utiliser pour manœuvrer des portes ou des barrières, ou bien pour réaliser le déplacement lent des aiguilles comme dans la réalité.

Coulisse automatisée à deux voies

Vous en savez assez sur les entrées et les sorties pour concevoir et réalisez votre premier automatisme. Il s'agit d'une coulisse automatisée constituée de deux voies 1 et 2, une aiguille d'entrée A et une aiguille de sortie B. Un signal à deux feux (vert-rouge) donne le signal de départ du train. Lorsqu'un train arrive, il est dirigé vers la voie libre et s'arrête car son signal est au rouge. Après une courte temporisation, l'autre train démarre dès que son signal est au vert, ce qui libère sa voie pour un autre train. La figure 23.2 représente cette coulisse **qui n'est parcourue que dans un seul sens**.

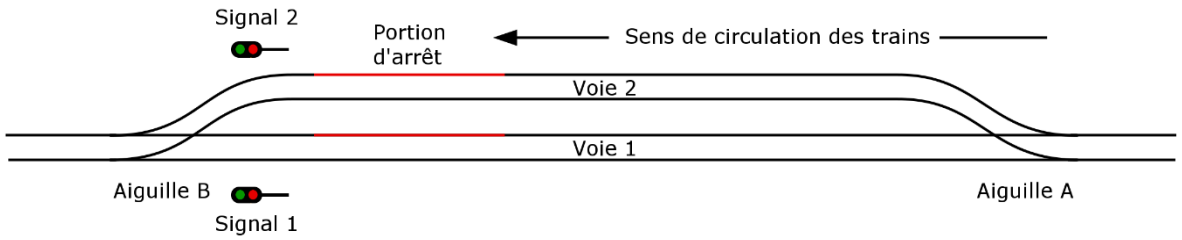

Figure 23. 2

Concevez les capteurs à positionner pour envoyer les informations d'entrée, voyez ce qu'il y a à positionner pour commander le réseau, et écrivez un programme fonctionnel pour gérer cette coulisse en alimentation analogique. Dessinez le schéma de câblage de l'automatisme. On admettra que les bobines de commande des aiguilles consomment moins de 500 mA et peuvent être commandées par un ULN2803.

L'automatisme qui vous a été proposé peut être monté sur un petit réseau comme un simple ovale avec deux voies sur un côté, ce que propose les coffrets de départ éventuellement complétés par un coffret additionnel comprenant deux aiguilles et quelques coupons de voie. Un train part de la voie 1, fait le tour du réseau et revient sur la voie 1. Quelques temps plus tard, un train part de la voie 2, fait le tour du réseau et revient sur la voie 2. Ce programme se répète autant qu'on le veut.

Essayons de décomposer cette tâche complexe en tâches élémentaires, comme nous l'avons fait pour changer les roues de notre voiture au chapitre 17. Voyons déjà quelles sont les conditions initiales à réaliser en début de programme.

Conditions initiales

Un train **arrêté** sur la voie 1, un train **arrêté** sur la voie 2.

Signal 1 et 2 au rouge.

Aiguille B vers voie 1 (le train 1 sort en premier).

Un capteur ILS est positionné en amont de l'aiguille A pour signaler l'arrivée d'un train dans la coulisse.

Voyons maintenant le déroulé du programme, c'est-à-dire l'enchainement de tâches élémentaires. Ce qui est en italique correspond à des commentaires. On supposera qu'un train met 10 secondes pour quitter sa voie et se retrouver sur le réseau en dehors de la gare ; ce temps est appelé **attente 1**. On supposera également que le train met 10 secondes pour réintégrer sa voie de stationnement à partir du moment où il passe au-dessus de l'ILS, et que l'autre train part 5 secondes après ce délai. La somme

des deux temps, soit 15 secondes, est appelée **attente 2**. Bien entendu, les valeurs des variables attente 1 et attente 2 sont réglables en fonction des besoins de chacun.

Programme

Mettre signal 1 au vert *(ce qui suppose auparavant d'éteindre le rouge sur ce signal)*

Alimenter portion d'arrêt voie 1

Le train 1 démarre

Attente 1 pour que le train 1 soit entièrement sorti

Mettre signal 1 au rouge

Retirer alimentation sur portion d'arrêt voie 1

Aiguille A vers voie 1 *(pour le retour du train)*

Attendre signal ILS

Au signal ILS, attente 2 *(pour que le train 1 soit entièrement rentré sur voie 1 + attente avant que train 2 démarre)*.

Le train 1 s'arrête sur voie 1

Fin d'attente 2 : mettre signal 2 au vert

Alimenter portion d'arrêt voie 2

Le train 2 démarre

Attente 1 pour que le train 2 soit entièrement sorti

Mettre signal 2 au rouge

Retirer alimentation sur portion d'arrêt voie 2

Positionner aiguille A vers voie 2

Attendre signal ILS

Au signal ILS, attente 2 *(pour que le train 2 soit entièrement rentré sur voie 2 + attente avant que train 1 démarre)*.

Le train 2 s'arrête sur voie 2

Fin d'attente 2

Revenir au début du programme.

Voyons maintenant quelles ressources sont nécessaires.

Ressources

4 sorties numériques pour les 4 LED des deux signaux.

1 entrée numérique pour le capteur ILS.

2 sorties numériques pour commander les deux relais 1RT alimentant les portions d'arrêt.

2 aiguilles à commander soit 4 bobines, ce qui représente 4 sorties.

Au total :

1 entrée et 10 sorties, ce qui est compatible avec un module Uno.

L'automatisme réalisé est ce qu'on appelle un **séquenceur** : le programme réalise des tâches les unes après les autres, toujours dans le même ordre et avec le même tempo. Il n'y a pas de test pour connaitre des conditions et prendre des décisions. Ce genre de programme est le plus simple à réaliser mais n'est pas toujours possible.

La sécurité est assurée par le signal donné par l'ILS ; en effet, si un train plante au cours de son circuit, l'ILS ne donnera aucun signal de retour et l'autre train ne démarrera jamais.

Comme il faut imaginer le pire, on peut penser que le train arrive sur l'ILS (ce qui démarre le compte à rebours du départ de l'autre train) puis plante sur l'aiguille A ; dans ce cas, l'autre train démarrera et viendra télescoper le train planté. On se rend compte qu'**un seul système de sécurité n'est pas suffisant pour garantir à 100% que les trains ne se rencontrent jamais** ; il en faut au moins deux, comme dans un avion où tous les équipements sont doublés, voire triplés. Mais si le circuit est équipé de cantons, cela marchera à 100%, car le deuxième train viendra s'arrêter sur le canton précédant celui du train planté.

Schéma de montage

La figure 23.3 donne le schéma de montage de notre application.

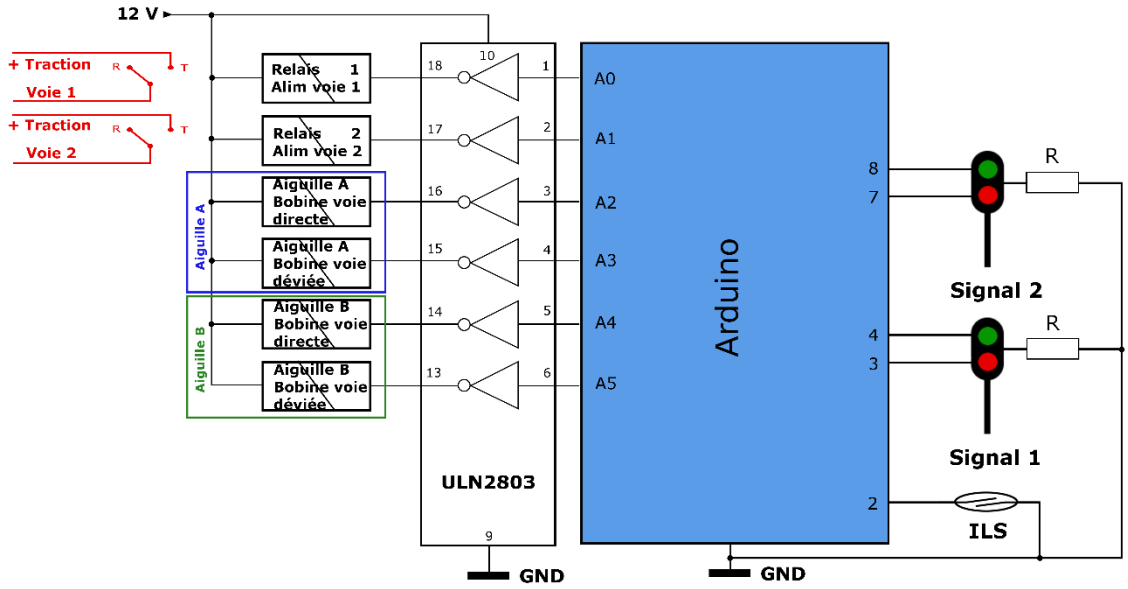

Figure 23. 3

Il y a six bobines à commander (deux pour les relais 1RT) et quatre pour les deux aiguilles ; nous utilisons pour cela les sorties analogiques A0 à A5 comme des sorties numériques. Comme nous l'avons conseillé, nous n'utilisons pas les sorties 0 et 1 (Rx et Tx) qui peuvent servir à la mise au point en affichant des données dans le moniteur pour contrôler la bonne exécution du programme principal. Nous réservons pour plus tard les sorties 5 et 6 pouvant faire de la PWM à 980 Hz pour une amélioration ultérieure du système (démarrage et freinage progressif). Les signaux lumineux (vert-rouge) sont à cathodes communes et sont commandés par un signal HIGH sur la broche. Les LED sont protégées par les résistances R (220 ou 330 Ohm). Le commun des aiguilles est relié au + d'une alimentation 12 V ; les bobines sont commandées par un signal bas en sortie de l'ULN2803, donc par un signal haut en entrée de l'ULN qui est la sortie d'Arduino.

Maquette de mise au point

Ceux qui n'ont pas de réseau peuvent fabriquer une maquette permettant de visualiser les différents courants que le module Arduino enverra vers les signaux, les relais et les aiguilles. Les anglo-saxons appellent cela un **mock-up**, ou si vous préférez : maquette, imitation.

La figure 23.4 montre cette maquette de mise au point où les relais et l'ULN2803 sont remplacés par des LED. Des LED vertes et rouges simulent les signaux, des LED jaunes simulent les relais d'alimentation des voies, et des LED blanches et bleues simulent les mouvements d'aiguille (LED blanche si aiguille directe et LED bleue si aiguille déviée). L'ILS est remplacé par un poussoir et l'entrée 2 est en mode INPUT_PULLUP.

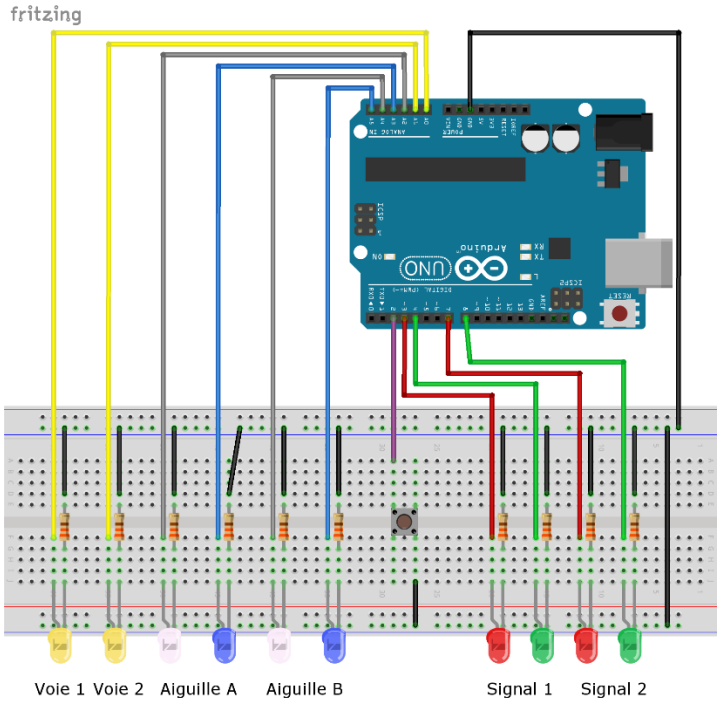

Figure 23. 4

L'allumage des différentes LED permet de contrôler que les signaux envoyés par Arduino respectent bien la séquence de gestion de la gare telle que nous l'avons conçue.

Programme associé

Le programme est un séquenceur, c'est-à-dire qu'il ne fait rien d'autre que de commander des sorties les unes après les autres. À un moment, il bouclera jusqu'à ce que l'ILS envoie son signal. Dès le signal reçu, il continuera son exécution pour traiter le train 2.

Un train met 10 s pour sortir de sa voie et se dégager de l'aiguille de sortie (aiguille B) : attente 1 est de 10 s.

Un train met 10 s pour venir s'arrêter à partir du moment où il passe l'ILS et le train d'à côté part 5 s après l'arrêt du train entrant : attente 2 est de 15 s.

La durée de manœuvre des aiguilles est de 1 s *(pour éviter que les bobines d'aiguille ne chauffent)*.

Le départ d'un train a lieu 1 s après la mise au vert du signal.

Commentaires du programme

Initialisation des variables :

gare_cachee_2voies

```
#define ILS 2
#define S1R 3
#define S1V 4
#define S2R 7
#define S2V 8
#define V1 14
#define V2 15
#define AA_dir 16
#define AA_dev 17
#define AB_dir 18
#define AB_dev 19

const int attente_0 = 1000;
const int attente_1 = 10000;
const int attente_2 = 15000;
```

#define ILS 2 indique au compilateur que chaque fois qu'il rencontre l'expression ILS, il doit la remplacer par 2 ; cela évite d'utiliser de l'espace mémoire puisqu'on ne fait pas appel à une variable.

Tous les #define permettent de définir comment est câblé le module ; il est en effet plus facile de faire le rapprochement entre AB_dir et Aiguille B bobine voie directe, plutôt que de se rappeler que cette bobine est branchée sur la sortie 18.

Attention : on ne met pas de point-virgule à la fin d'une ligne utilisant #define ! Et on ne met pas non plus de signe égal.

Les délais d'attente sont des constantes (ces délais ne varient pas au cours du programme). De plus, ce sont des nombres entiers. En utilisant le mot clé const, on ne gaspille pas la place en mémoire. Comparez la place mémoire utilisée si on utilise le mot const et si on ne l'utilise pas en déclarant ces constantes simplement avec int (pour entier).

À la place des #define, on aurait pu aussi définir des constantes ; par exemple, #define ILS 2 peut être remplacé par const byte ILS = 2 ;

On peut faire la même chose pour S1R, S1V, etc. mais il ne faut pas oublier le point-virgule à la fin de chaque instruction dans ce cas.

Le cours sur les microcontrôleurs de l'Ecole Polytechnique Fédérale de Lausanne fait un grand emploi des #define, mais le site Arduino conseille plutôt d'utiliser les définitions de constantes (voir à ce sujet la page du site Arduino définissant #define).

Fonction setup :

gare_cachee_2voies

```
void setup() {
  // entree ILS en mode pull-up interne
  pinMode (ILS, INPUT_PULLUP);
  // sorties pour signaux (S1R signifie Signal 1 Rouge)
  pinMode (S1R, OUTPUT);
  pinMode (S1V, OUTPUT);
  pinMode (S2R, OUTPUT);
  pinMode (S2V, OUTPUT);
  // sorties pour relais d alimentation des voies (V1 est voie 1)
  pinMode (V1, OUTPUT);
  pinMode (V2, OUTPUT);
  // sorties pour solenoides aiguilles (AB_dev pour Aiguille B deviee)
  pinMode (AA_dir, OUTPUT);
  pinMode (AA_dev, OUTPUT);
  pinMode (AB_dir, OUTPUT);
  pinMode (AB_dev, OUTPUT);
  // conditions initiales : deux trains arretes
  digitalWrite (V1, LOW);
  digitalWrite (V2, LOW);
  digitalWrite (S1V, LOW);
  digitalWrite (S2V, LOW);
  digitalWrite (S1R, HIGH);
  digitalWrite (S2R, HIGH);
}
```

L'entrée 2 est initialisée en choisissant le mode INPUT_PULLUP, ce qui permet de monter l'ILS entre l'entrée et la masse. Lorsque l'ILS est fermé, l'entrée reçoit un signal LOW.

Toutes les autres broches sont initialisées en sorties.

Les conditions initiales sont deux trains arrêtés, donc voies 1 et 2 non alimentées, et signaux 1 et 2 au rouge (c'est-à-dire vert éteint et rouge allumé).

Fonction loop :

gare_cachee_2voies

```
void loop() {
  // Aiguille B vers V1
  digitalWrite (AB_dir, HIGH);
  delay (attente_0);
  digitalWrite (AB_dir, LOW);
  delay (attente_0);
  // signal 1 passe au vert
  digitalWrite (S1R, LOW);
  digitalWrite (S1V, HIGH);
  delay (attente_0);
  // depart train 1
  digitalWrite (V1, HIGH);
  delay (attente_1); // train parti
  // signal 1 passe au rouge
  digitalWrite (S1V, LOW);
  digitalWrite (S1R, HIGH);
  delay (attente_0);
  // voie 1 non alimentee
  digitalWrite (V1, LOW);
  delay (attente_0);
  // positionnement aiguille d'entree pour retour
  digitalWrite (AA_dir, HIGH);
  delay (attente_0);
  digitalWrite (AA_dir, LOW);
  delay (attente_0);
  while (digitalRead (ILS) == HIGH) { // attendre retour du train 1
  }
  // le train 1 passe sur l'ILS
  delay (attente_2); // train 1 arrete depuis 5 s
  // aiguille B vers V2
```

<-Début

gare_cachee_2voies

```
  // aiguille B vers V2
  digitalWrite (AB_dev, HIGH);
  delay (attente_0);
  digitalWrite (AB_dev, LOW);
  delay (attente_0);
  // signal 2 passe au vert
  digitalWrite (S2R, LOW);
  digitalWrite (S2V, HIGH);
  delay (attente_0);
  // depart train 2
  digitalWrite (V2, HIGH);
  delay (attente_1); // train parti
  // signal 2 passe au rouge
  digitalWrite (S2V, LOW);
  digitalWrite (S2R, HIGH);
  delay (attente_0);
  // voie 2 non alimentee
  digitalWrite (V2, LOW);
  delay (attente_0);
  // positionnement aiguille d'entree pour retour
  digitalWrite (AA_dev, HIGH);
  delay (attente_0);
  digitalWrite (AA_dev, LOW);
  delay (attente_0);
  delay (attente_0);
  while (digitalRead (ILS) == HIGH) { // attendre retour du train 2
  }
  // le train 2 passe sur l'ILS
  delay (attente_2); // train 2 arrete depuis 5 s
  // recommence la sequence
}
```

<- Suite

Fin du programme

Un départ de train commence toujours par positionner l'aiguille de sortie, mettre le signal au vert, alimenter la voie.

L'attente 1 est suffisamment longue pour que le train soit complètement sorti de la voie. On remet ensuite le signal au rouge (et pour cela il faut penser avant à éteindre le vert), puis on coupe l'alimentation sur la portion d'arrêt pour que le train s'arrête lorsqu'il reviendra.

On positionne ensuite l'aiguille d'entrée.

On attend le retour du train, donc on attend le signal de l'ILS positionné en amont de l'aiguille A (avec la boucle while, le programme boucle à ce point jusqu'à ce que le signal de l'ILS soit LOW).

À partir du moment où le signal est détecté, l'attente 2 permet au train de rentrer en voie, de s'arrêter et donne 5 secondes de plus avant de commander le départ du train 2.

La séquence est la même pour le train 2 avec ses propres signaux et aiguilles.

Lorsque le train 2 est arrêté depuis 5 secondes, on recommence le programme, donc on fait repartir le train 1.

Contrôle sur la maquette

Voici dans l'ordre ce qu'on doit observer sur notre maquette :

Signaux 1 et 2 rouges, voies 1 et 2 LED jaune éteinte.

Aiguille B : LED blanche pendant 1 s

Signal 1 passe au vert

LED jaune voie 1 s'allume pendant (attente 1)

Signal 1 passe au rouge

Une seconde après, LED jaune voie 1 s'éteint

Aiguille A : LED blanche pendant 1 s

On appuie sur poussoir pour simuler retour du train 1

Après (attente 2) deux trains arrêtés, il faut faire partir le train 2.

Aiguille B : LED bleue pendant 1 s

Signal 2 passe au vert

LED jaune voie 2 s'allume pendant (attente 1)

Signal 2 passe au rouge

Une seconde après, LED jaune voie 2 s'éteint

Aiguille A : LED bleue pendant 1 s

On appuie sur poussoir pour simuler retour du train 2

Après (attente 2) deux trains arrêtés.

Etc.

Alors, vous pensez que ça va fonctionner ?

Vérifiez-le en cliquant ICI.

https://vimeo.com/153538422

Vous pouvez maintenant améliorer ce montage et gérer une gare constituée de quatre ou cinq voies.

Vous avez vu que concevoir un automatisme n'est pas très difficile en électronique programmée ; on réfléchit bien au problème, on évalue les ressources nécessaires pour voir si la solution est compatible avec ce que peut proposer le module Uno (sinon on choisit un autre module plus puissant comme le Mega2560), on pense à la sécurité de circulations des trains pour éviter qu'ils ne se rencontrent, on décompose le problème en tâches élémentaires, on écrit le programme et on le teste soit grandeur nature sur le réseau, soit avec un mock-up. C'est exactement comme cela que nous avons opéré.

À retenir sur la commande du réseau :

- Commander un réseau revient à commander des portions de voie souvent grâce à des relais ou des transistors.
- Les mouvements des animations du réseau sont réalisés avec des moteurs à courant continu, des moteurs pas à pas ou des servos.
- Il existe des bibliothèques pour gérer les moteurs pas à pas et les servos.
- Un problème complexe comme une gare à deux voies est résolu en le décomposant en tâches élémentaires.
- Il est nécessaire d'évaluer les ressources du microcontrôleur ou du module Arduino.
- Dans un automatisme, il faut penser à la sécurité pour éviter que les trains ne se rencontrent.
- On peut tester un programme sur une maquette très simple où des LED remplacent et simulent des relais ou des moteurs.

Liste des composants nécessaires pour :

La gare cachée à deux voies :

Un circuit intégré ULN 2803

Deux relais 1 RT

Un ILS

Deux DEL vertes et deux DEL rouges (ou deux signaux à deux feux vert-rouge à cathodes communes)

Deux résistances de 330 Ω

Le mock-up « Gare cachée à deux voies » :

DEL : deux vertes, deux rouges, deux bleues, deux blanches et deux jaunes

Dix résistances de 330 Ω

Un bouton poussoir