Master 1 Miage CL
CSI - Développement Logiciel

Université de Nantes
2017/2018

Feuille de travaux pratiques n° 2
Projet Riley Rover

Etude de cas

Ce document présente le cas d'étude et une première analyse en UML. On s'intéresse à un robot mobile autonôme à contrôler à distance.

1 Le cas d'étude en bref

Le cas porte sur un équipement domotique simplifié, la gestion d'une porte de garage en considérant deux dispositifs, un *client* sous forme de télécommande ou de smartphone et un *serveur*, le logiciel de contrôle de la porte, qui pilote les différents dispositifs physique. Une description est donnée en annexe **??**.

Il s'agit de concevoir et mettre en œuvre un logiciel de gestion de portes, dont l'analyse du domaine et les spécifications des exigences logicielles (fonctionnelles et non fonctionnelles) sont fournies.

1.1 Système et périmètre

Le système en fonctionnement comprend des dispositifs matériels et logiciels pour un véhicule autonome [1]

> Dans un souci d'évolution, nous donnons une description plus abstraite que celle du modèle d'implantation mais moins complexe que celle d'un vrai véhicule autonome (*cf.* figure 1).

1.2 Mise en œuvre

Le prototypage se fera en utilisant des automates LEGO MINDSTORMS EV3, un ensemble d'outil de construction, programmation et de commande de robots LEGO.

La maquette s'inspirera de celles proposées dans le document `RileyRover_BI.pdf` par Damien Kee [2] et disponible sur `http://www.damienkee.com/home/2013/8/2/rileyrover-ev3-classroom-robot-design.html`

Le système est organisé en trois parties :

— un véhicule, représenté par un système réactif avec un contrôleur, des capteurs et des actionneurs pour agir sur le système électromécanique support du véhicule. Seule la partie logicielle nous importe.
— un panneau de commande mobile pour interagir avec le véhicule et
— le contexte, qui est considéré comme passif, en ce sens que seuls les capteurs permettent d'en obtenir des informations, il n'y a pas de communication explicite avec l'environnement en dehors du panneau de commande.

> Pour l'aspect recherche, la documentation des modèles et le code seront écrits en anglais.

1.3 Modélisation

Nous donnons ici ne analyse du système c'est à dire une description du système à travers des modèles UML.

1. `https://www.techniques-ingenieur.fr/actualite/dossier/transport-vehicule-autonome/#dessin-du-mois`
2. voir aussi `http://www.damienkee.com/activites-de-classe-ev3/`

source : https://aruco.com/2014/07/voiture-autonome-2015/

Figure 1 : *Véhicule autonome vs Base du robot Riley Rover*

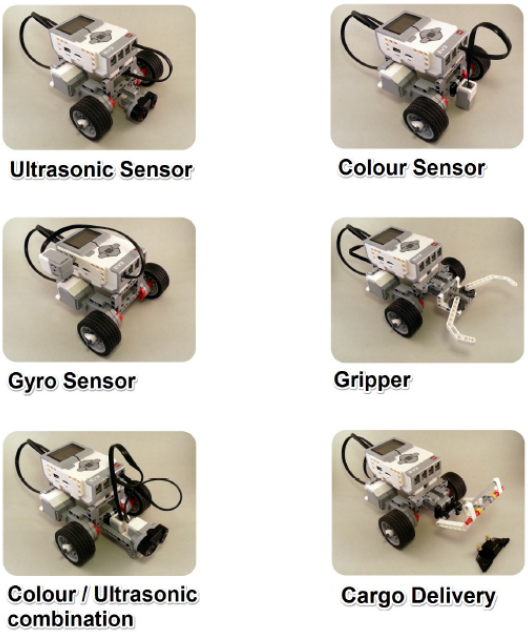

Figure 2 : *Variantes du robot Riley Rover*

1.4 Système et périmètre

La structure matérielle est composée de deux parties :

— un véhicule (robot) mobile sur lequel on place constituée d'un contrôleur, de capteurs, de moteurs et de capteurs divers et variés (détecteur d'obstacle distant, détecteur de contact, détecteur de luminosité, gyroscope pour détecter rotations et orientation).

— un panneau de commande mobile (ou télécommande) permettant de donner des instruction au robot : démarrer/arrêter, avancer/reculer, tourner à gauche/à droite, monter/descendre, avertisseur sonore...

Dans une vision itérative, le périmètre sera revu (agilité). Il s'agit ici d'un premier modèle, il sera à améliorer lors des différentes itérations.

Dans une vision incrémentale, nous prévoyons d'augmenter le périmètre en ajoutant un type de capteur à chaque itérations, on augmente ainsi progressivement la complexité.

1. Moteur de poussée horizontale (le véhicule avance ou recule)
2. Capteurs de contact (le véhicule réagit à un contact - arrêt ou mouvement inversé)
3. Capteur de distance (ultra-son en lego) on détecte les obstacles pour réagir)
4. Capteur de luminosité (couleurs en légo) on adapte la vitesse à la visibilité.
5. Gyroscope, on adapte la position au mouvement en cours (devant = 0, derrière, à gauche, à droite)[a].

On distingue aussi le comportement normal des incidents et pannes, voire accidents.

a. `https://www.youtube.com/watch?v=S5SjhfVPbNs`

Enfin on pourra ultérieurement ajouter des fonctions "métiers au véhicule : pousser des éléments, prendre et déposer des objets, transporter et suivre une trajectoire - c'est alors un Véhicule à guidage automatique ou AGV…

Dans une version abstraite du diagramme de classes, figure 3, nous avons uniquement des opérations.

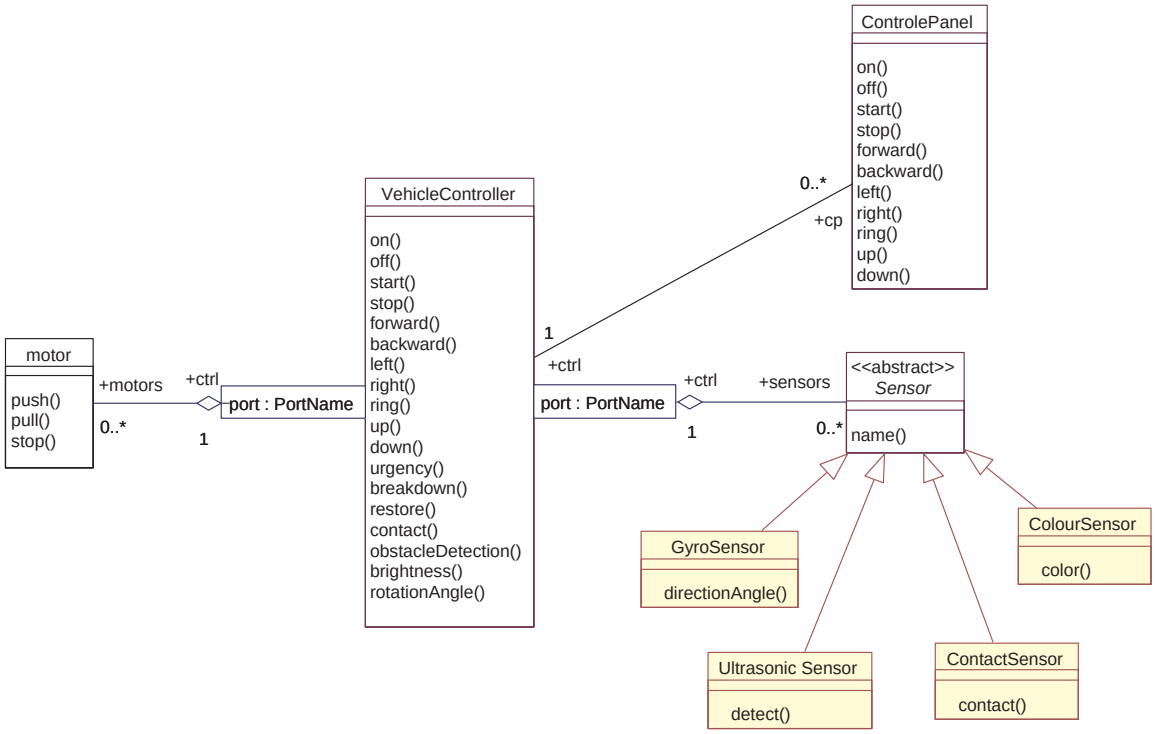

Figure 3 : *Diagramme de classes générique - Vehicle*

Nous pourrons, par la suite (durant la phase de conception) ajouter éventuellement un attribut `état` dans chacune des classes ayant un comportement dynamique (i.e. représenté par un diagramme états-transitions), cet attribut servant de support d'enregistrement de l'état courant. Cela risque toutefois de faire double emploi avec un attribut du méta-modèle. De la même façon,nous pourrions ajouter un attribut pour gérer l'historique.

Nous avons abstrait la relation entre le contrôleur et ses dispositfs par deux agrégations qualifiées, ce n'est pas faux en terme de représentation structurelle de l'EV3 mais évidemment les API et protocoles de communications

entre le contrôleur et ses dispositifs est fonction de la nature du dispositif, ce qui donne plutôt le modèle de la figure 4 du point de vue de la communication.

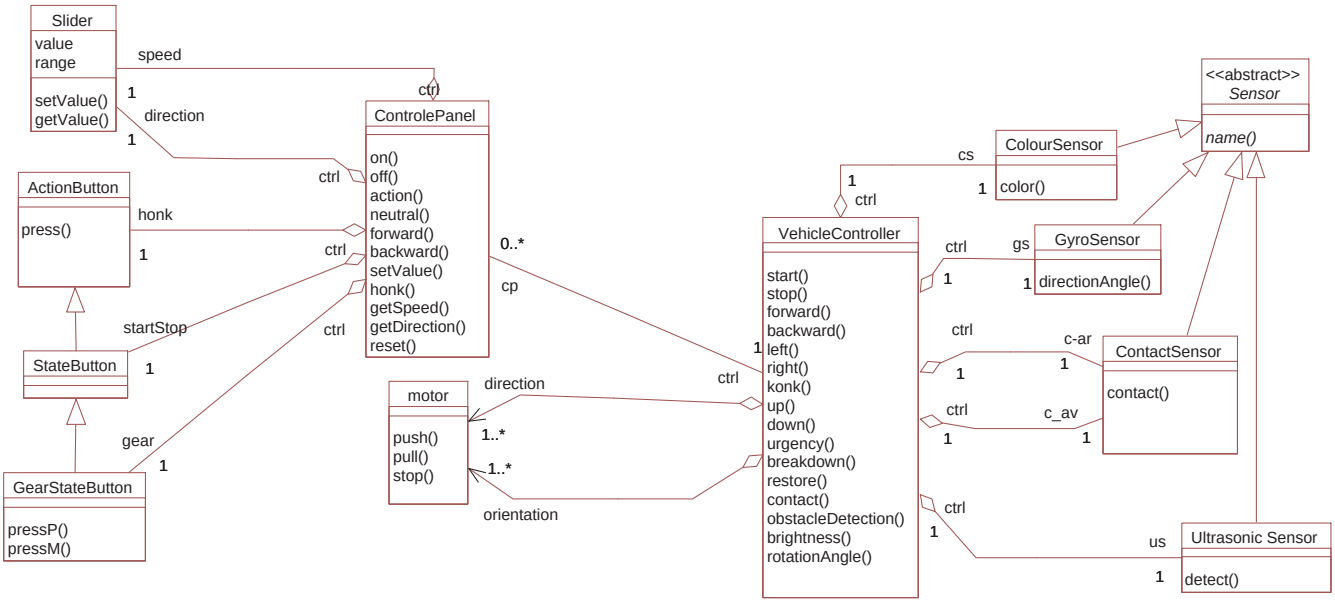

Figure 4 : *Diagramme de classes - Vehicle*

2 Description du fonctionnement et des composants

Ce système comprend un véhicule qui bouge (avancer, reculer, tourner, monter ou descendre), des moteurs pour actionner les mouvement du véhicule (tirer, pousser) et des capteurs pour recueillir des informations (contact, détection obstacle, luminosité, direction). Les capteurs sont de deux types : les capteurs actifs qui signalent à leur contrôleur un changement d'état (exemple : un capteur de contact signale qu'un contact a eu lieu) et des capteurs passifs qui fournissent une information du contexte à la demande du contrôleur (exemples : le gyroscope fournit un angle, le capteur de couleur fournit une couleur, le capteur à ultra-sons donne une distance...).

L'interaction avec l'usager se fait par un panneau de contrôle qui dispose ainsi d'une télécommande pour piloter le véhicule avec simplement des boutons d'action : marche, arrêt, avancer, reculer, stop, tourner à gauche ou droite, avertisseur sonore, accélérer, ralentir.... On peut aussi imaginer d'autres actions ou d'autres retours visuels.

Le principe de fonctionnement du système est le suivant. Supposons le véhicule arrêté. L'usager met en route le système (bouton `on/off`) puis démarre le véhicule (bouton `start/stop`) puis enclenche la marche avant du véhicule en appuyant sur le bouton `forward` de sa télécommande. Il peut arrêter le véhicule en réappuyant sur le bouton `start/stop`, le moteur s'arrête. Dans le cas contraire le véhicule avance jusqu'à rencontrer un obstacle qui déclenche le capteur de contact avant `c_av` qui entraîne l'arrêt du moteur. Appuyer sur le bouton `backward` enclenche la marche arrière du véhicule s'il est démarré.

On peut arrêter le véhicule en appuyant sur le bouton `Stop`, le moteur s'arrête. À tout moment, si quelqu'un appuie sur un bouton d'arrêt d'urgence alors le véhicule se bloque. Pour le réactiver (dans le même état) on active le mécanisme de reprise `Restore`.

On peut imaginer un système de vitesse plus complexe pour la marche avant, en appuyant plusieurs fois sur le bouton `forward` on passe les vitesses.

2.1 Panneau de commande

Le panneau de commande (ou télécommande) dispose de différents boutons et d'indicateurs. Il s'agit d'une vision des choses et l'IHM n'est pas détaillée ici, elle est libre à ce stade.

Hypothèse 1 (contexte) *Il s'agit d'une vision des choses et l'IHM n'est pas détaillée ici, elle est libre à ce stade.*

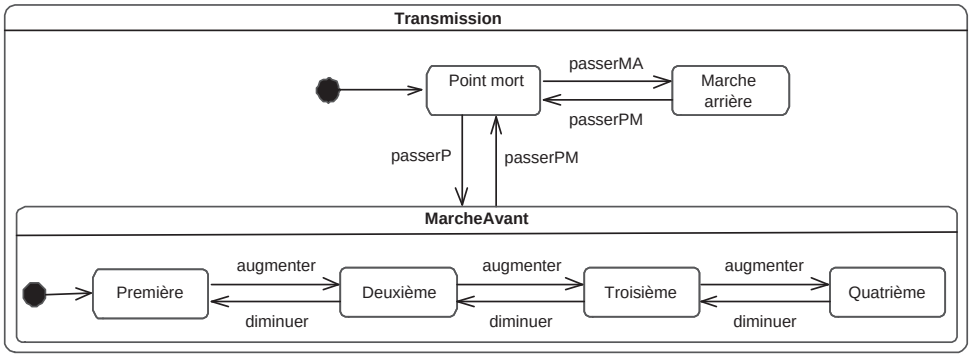

Figure 5 : *Diagramme Etats-transitions de la transmission - Vehicle*

Les boutons, sur lesquels l'utilisateur peut appuyer, représentent des déclenchements d'ordres utilisateur. Parfois nous avons groupé deux actions sur un même bouton pour réduire les contrôles de cohérence au sein du contrôleur. A la base, appuyer sur un bouton de l'interface envoie simplement un message au contrôleur. Nous donnons parfois une interprétation qui dépend plus de l'intention car c'est le contrôleur qui donnera le sens exact.

— Bouton mémoire `start/stop`. Une pression sur le bouton `start/stop` déclenchera la mise en route du véhicule et en appuyant à nouveau on éteint.
— Bouton mémoire à trois valeurs et deux actions `gear` *forward/neutral/backward* (FNB). Une pression sur le bouton `plus` fait avancer le véhicule (s'il est au point mort `neutral`) et en appuyant sur le bouton `minus` on se trouve au point mort. Une pression sur le bouton `minus` fait reculer le véhicule (s'il est au point mort `neutral`) et en appuyant sur le bouton `plus` on se trouve au point mort.
— Bouton action `honk`. Une pression sur le bouton `honk` active le klaxon (*horn*).

On peut avoir un panneau de commande plus spécifique à contrôleur (plus "smart") qui masquerait les boutons non activables dans l'état courant du contrôleur.

Deux curseurs permettent de donner l'intensité d'une action

— Le curseur `vitesse` donne la puissance des moteurs de traction sur une échelle de 1 à 10.
— Le curseur `direction` donne l'angle d'inclinaison sur une échelle de -90 à +90.

Des outils de visualisation des indicateurs de fonctionnement seront fournis, leur présentation est laissée à l'appréciation des développeurs.

L'interprétation des commandes et ordres du panneau est faite par le contrôleur en fonction de l'état du véhicule et des événements détectés par les capteurs. La télécommande, seule, ne peut pas décider de la marche à suivre. La télécommande, lorsqu'elle est activée, réagit aux événements (appui sur un bouton ou variation d'un curseur) et, à chaque fois, se *contente* de signaler au contrôleur qu'il y a eu pression sur le bouton ou variation d'une intensité.

Dans ce qui suit, la variable `ctrl` représente le contrôleur, le panneau étant le contrôleur des boutons ou contrôleur composite.

Le comportement d'un bouton action est modélisé dans la figure 6.

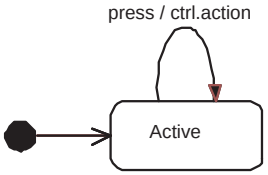

Figure 6 : *Diagramme Etats-transitions du panneau de commande - ActionButton*

Le comportement d'un bouton mémoire est modélisé dans la figure 7.

Le comportement d'un bouton mémoire à trois valeurs est modélisé dans la figure 8. Nous utilisons ce bouton pour la visualisation mais un bouton action aurait suffit, le contrôleur gérant alors l'état du boitier de vitesse.

Le comportement d'un curseur est modélisé dans la figure 9.

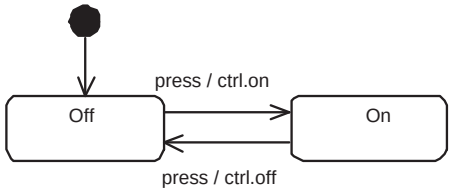

Figure 7 : *Diagramme Etats-transitions du panneau de commande - StateButton*

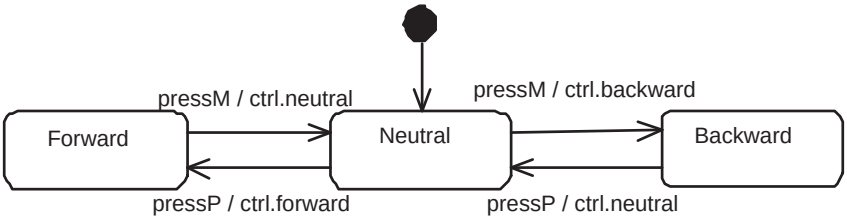

Figure 8 : *Diagramme Etats-transitions du panneau de commande - GearStateButton*

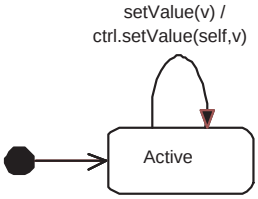

Figure 9 : *Diagramme Etats-transitions du panneau de commande - Slider*

Le comportement du panneau est une composition des comportements des boutons.

```
context ControlPannel::action(bt)
   pre: true
   post: if bt=honk then self.ctrl.honk
context ControlPannel::on(bt)
   pre: true
   post: if bt=startStop then self.ctrl.start()
context ControlPannel::off(bt)
   pre: true
   post: if bt=startStop then self.ctrl.stop()
context ControlPannel::neutral(bt)
   pre: true
   post: if bt=gear then self.ctrl.neutral()
context ControlPannel::forward(bt)
   pre: true
   post: if bt=gear then self.ctrl.forward()
context ControlPannel::backward(bt)
   pre: true
   post: if bt=gear then self.ctrl.backward()
context ControlPannel::setValue(sl,v)
    -- to refine with the real device API
    -- vspeed is a function that computes the true speed value
   pre: v <> 0
   post: if sl=speed then
         if self.speed < v the self.ctrl.up(vspeed(v-self.speed))
         else self.ctrl.down(vspeed(self.speed-v))
context ControlPannel::setValue(sl,v)
    -- to refine with the real device API
    -- vangle is a function that computes the true angle value
   pre: v <> 0
```

```
post: if sl=direction then
      if v < 0 the self.ctrl.left(vangle(-v))
      else self.ctrl.right(vangle(v))
```

Les éléments fournis ici sont intuitifs, ils peuvent êtres traités directement par le contrôleur du véhicule. Cela fait partie des API de communication à mettre au point.

2.2 Capteur

Il y a différents types de capteurs. Les capteurs actifs signalent un changement d'état. Les capteurs passifs sont interrogés. Seul les capteurs actifs nécessitent un diagramme état transition.

2.2.1 Capteur de contact

Le capteur de contact est actif, signale à son contrôleur lorsque le contact a lieu. Il se comporte donc de façon comparable au panneau de commande et nous pouvons modéliser son comportement par la figure 10.

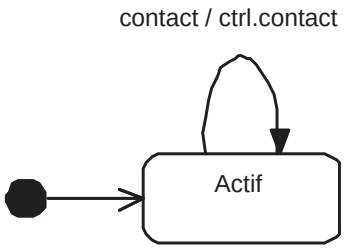

Figure 10 : *Diagramme Etats-transitions de la télécommande - Vehicle - Capteur*

2.3 Moteur

Le moteur fait bouger le véhicule. En pratique on peut avoir plusieus moteurs de traction (avancer/reculer) mais aussi des moteurs de direction (gauche/droite). Quelle que soit sa fonction, soit un moteur pousse (`push`), tire (`pull`)... soit il est à l'arrêt. Nous avons donc trois états, `Push`, `Pull`, `Stopped`. Le résultat de ces actions est mentionné en filigrane dans le texte. Lorsque le moteur pousse, le véhicule avance ; lorsqu'il tire, le véhicule recule. Nous supposons que le moteur réalise trois actions : `push`, `pull` et `stop`. Ces actions sont exécutées lorsque le contrôleur lui demande de le faire. Son comportement est alors celui de la figure 11.

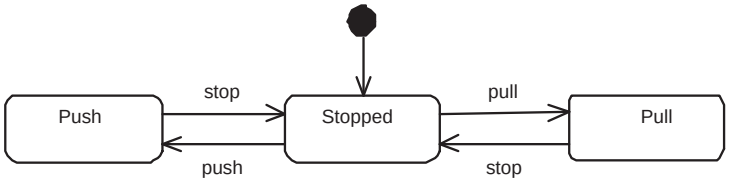

Figure 11 : *Diagramme Etats-transitions du moteur - Vehicle*

2.4 Véhicule et contrôleur

Le comportement observable du véhicule sera assimilée à celui de son contrôleur qui en représente finalement le jumeau virtuel (*digital twin*). Sa modélisation va faire appel à plusieurs notions. Une première version du comportement du véhicule est décrite par la figure 12.

Le contrôleur reçoit les signaux des capteurs et de commande et pilote le véhicule. Son rôle est d'analyser ces signaux et de décider, en fonction de ceux-ci, ce que doit faire le véhicule. Pour décrire ceci, nous choisissons

d'adopter la même structure que celle utilisée pour décrire le comportement des dispositifs. Les états du contrôleur correspondent, en fait, aux différentes étapes dans le fonctionnement du véhicule.

Au niveau macroscopique, le véhicule est en mode `manuel` ou `assisté`. Lorsqu'il est en service assisté, il peut être dans un des trois états principaux suivants : point mort, en marche avant ou en marche arrière. Il se bloque dès que le contrôleur le décide ou qu'un obstacle est rencontré (`Contact`). Cet arrêt dure jusqu'à ce qu'on redémarre avec le message `start` (par le panneau de commande).

L'état `Assistance` est en fait un super-état (il regroupe `Neutral`, `Contact` et `Moving`). Ce dernier est lui-même un super-état (il regroupe `Forward`, `Backward` et `Moving`). De fait le diagramme états-transitions est un automate hiérarchique. Les super-états permettent de factoriser les transitions entrantes et sortantes. Par défaut, la transition entrante va dans l'état initial lorsqu'il existe et effectue les actions d'entrée : vérification des capteurs et actionneurs..

En sortie de `Assistance`, quel que soit le sous-état, o arrête le moteur.

Lorsque le véhicule est en déplacement (état `Moving`), on peut accélérer, ralentir, tourner à gauche ou à droite, selon les règles de fonctionnement de ces opérations.

La variable `mtD` correspond aux moteurs de direction qui commande les mouvement de traction. La variable `mtO` correspond aux moteurs d'orientation qui commande les mouvement de direction gauche-droite.

Un historique est utile pour gérer la reprise du fonctionnement après blocage urgent. Un véhicule bloqué à un endroit doit, après déblocage, reprendre son mouvement. Lors de la reprise, il reprend son activité là où elle s'était arrêtée.

Le diagramme d'activités de la figure 12 décrit ce comportement. La variable `pt` représente la porte.

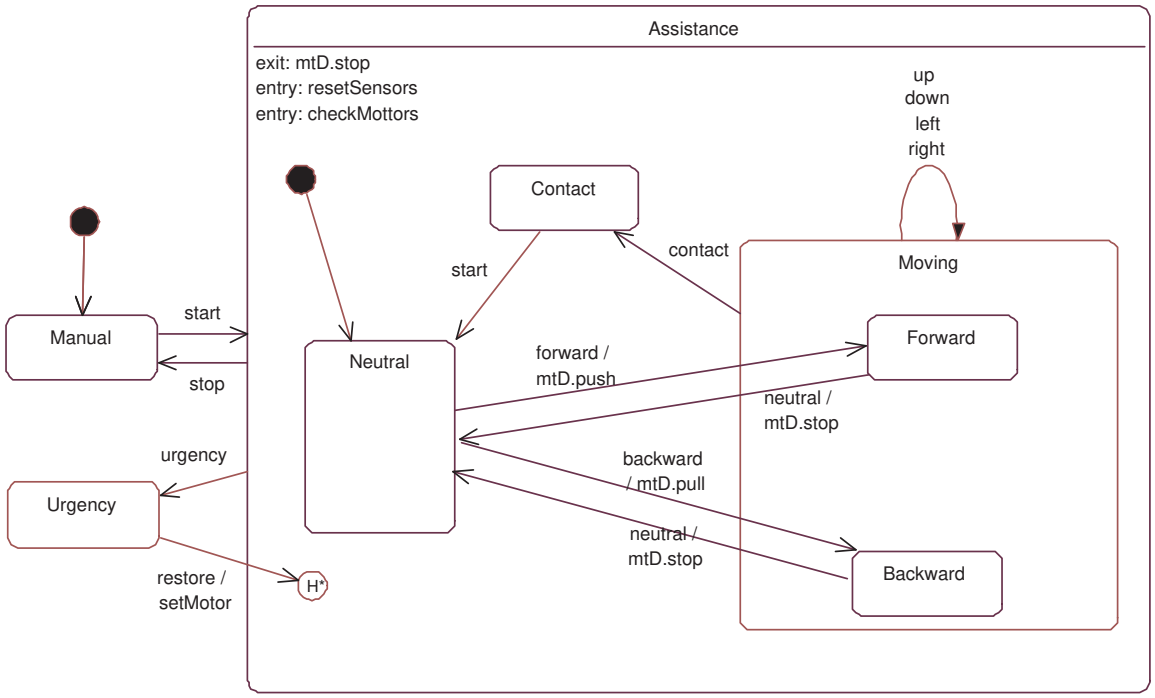

Figure 12 : *Diagramme Etats-transitions du contrôleur - Porte*

> Il s'agit d'une première version, nous pouvons y ajouter des aspects complémentaires pour le traitement des obstacles, la luminosité et les contraintes de sécurité.

2.5 Protocole de communication

Les dispositifs physique échangent directement avec le contrôleur. On suppose un système simple d'échanges de messages en circuit fermé. La télécommande communique à distance avec le contrôleur pour commander l'ouverture et la fermeture des portes. En pratique, la comunication avec LEGO MINDSTORMS EV3 se fait en bluetooth ou Wifi. En extension, un protocole sécurisé vérifie l'authenticité de la télécommande pour ces échanges.

3 Exigences, contraintes et hypothèses

Nous précisons ici quelques exigences du système.

3.1 Hypothèses et restrictions

On ne traite que de la partie logicielle. On ne tient pas compte du fonctionnement en mode manuel, lorsque le contrôleur est déconnecté.

La modélisation proposée ici reste une esquisse, tous les éléments n'ont pas été analyses en détail (notamment les interfaces des dispositifs matériels, les communications et protocoles associés...). Malgré tout, le modèle proposé permet de mieux comprendre le système et d'en faire une simulation, non seulement logicielle (le *twin digital*) mais aussi matérielle avec le prototype Lego.

- Pour le prototypage, on représentera le système d'une manière propre aux dispositifs disponibles en Lego Mindstorm EV3 : la vitesse, les angles, la direction seront représentée par un degré de rotation sur un bras armé par le grand moteur.
- Les signaux sonores sont déclenchés par le boitier EV3 lui-même.
- Le capteur de distance (détection d'obstacles) sera mis en œuvre par le capteur d'ultra-sons.
- Le capteur de mouvement (angles de la direction) sera mis en œuvre par le gyroscope.
- Le capteur de luminosité sera mis en œuvre par le détecteur de couleur : clair pour le jour et sombre pour la nuit.

3.2 Exigences fonctionnelles

Le système doit permettre de manipuler la lumière via son contrôleur. C'est un système distribué sur plusieurs dispositifs physique, la phase de mise en route et d'initialisation n'est donc pas triviale, de même que la phase d'arrêt.

Les exigences fonctionnelles sont classées par priorité.

1. Gestion de configuration : lancement, arrêt des différents dispositifs.
   (a) Initialiser le mode automatique, paramétrer
      i. Vérifier l'état des dispositifs physique (capteurs et actionneurs), panneau de commande, supervision.
      ii. Initialiser chaque dispositif.
      iii. Initialiser le mode automatique du contrôleur.
      iv. Vérifier la cohérence globale (assertions).
   (b) Basculer en mode Manuel
2. Gestion des panneaux de commande (écran PC ou mobile)
   (a) Enregistrer un panneau
   (b) Supprimer un panneau
3. Gestion des modes et mouvements de base (via le panneau)
   (a) Allumer / Eteindre
   (b) Avancer / Reculer
4. Gestion des obstacles (via les capteurs)
   (a) Toucher (capteur de contact
   (b) S'approcher (via les détecteurs de distance)
5. Gestion des mouvements de direction et de la vitesse (via le panneau et gyroscope)
   (a) Gauche / Droite
   (b) Accélérer/ralentir
6. Monitoring
   (a) Afficher l'état du système :
      i. Etat du contrôleur,
      ii. Etat des dispositifs matériels

(b) Gérer les événements
    i. luminosité extérieure,
    ii. détection de présence
(c) Gestion d'un historique (Log)
    i. Liste des commandes et des états du contrôleur
    ii. Liste des anomalies et changements de modes (manuel/automatique)
    iii. Journalisation des événements et états pour tracer la source d'erreurs

7. Gestion des incidents
   (a) Passer en mode Manuel
   (b) Revenir en mode Automatique
   (c) Prise en compte des interruptions de courant et des opérations de reprise.
   (d) Prise en compte des communications avec les dispositifs matériels et des opérations de reprise.
   (e) Suivi des pannes (journalisation)
8. Divers.

3.2.1 Communication et prototypage

En pratique, la comunication avec LEGO MINDSTORMS EV3 se fait en bluetooth ou Wifi pour la commande et en liaison filaire pour le reste.

En extension, on définira un protocole sécurisé vérifiant l'authenticité de la télécommande pour ces échanges.

3.3 Extra-Fonctionnelle

Le système doit permettre de manipuler le véhicule en toute sécurité.

— La porte ne peut-être à la fois ouverte totalement et fermée totalement.
— L'état des dispositifs est cohérent avec celui du contrôleur (cohérence des capteurs).
  — Si le véhicule est en mouvement, le contrôleur la perçoit comme tel.
  — Si le véhicule est arrêté, le moteur est à l'arrêt.
  — Si le véhicule avance, le moteur de traction est en poussée.
  — Si le véhicule recule, le moteur de traction est en tirage.
  — Si le véhicule tourne à gauche, le moteur de direction est en poussée.
  — Si le véhicule tourne à droite, le moteur de direction est en tirage. . . .
— En cas de dysfonctionnement d'un dispositif, le véhicule se met automatiquement en mode manuel, par exemple si un capteur n'est plus actif (ne répond plus).
— Lorsqu'un obstacle est détecté en mouvement, le véhicule se bloque. Lorsqu'il reprend son mouvement, aucun obstacle ne doit être détecté.
— En cas de rupture de communication avec le panneau de commande (pas de nouvelle action au bout d'un certain temps), le véhicule se met automatiquement en mode manuel.
— . . .

3.4 Exigences techniques

La mise en œuvre du prototype se fera avec les outils Java pour Lego Mindstorm, la plate forme **Lejos**.

3.5 Evolutions à prévoir

On peut imaginer des facilités supplémentaires paramétrables.

— Un détecteur de présence empêche le véhicule d'avancer ou reculer en cas d'obstruction, in signal sonore est émis. Lorsqu'un obstacle est détecté en mouvement, le véhicule ralentit, klaxonne ou se bloque. Lorsqu'il reprend son mouvement, aucun obstacle ne doit être détecté.
— La porte "au point mort" s'arrête au bout 2 minutes.
— La porte se met en demi-ouverture.