Projet de Programmation synchrone

M2 Informatique UP & 3A EIDD

2022–2023

1 Introduction

Les langages synchrones sont utilisés pour développer des logiciels de contrôle-commande dans de nombreux domaines industriels, notamment dans le cadre de systèmes critiques. Le but de ce projet est de vous donner une expérience préliminaire de ce genre de développement, au delà des petits programmes que vous avez écrits lors des séances de travaux pratiques de la première moitié du semestre. Il s'agira pour vous d'écrire un programme synchrone contrôlant un dispositif physique simplifié mais non-trivial. Nous évaluerons sa performance à l'aide d'un simulateur lui aussi écrit sous la forme d'un programme synchrone, et qui sera mis à votre disposition afin de permettre l'auto-évaluation.

Le dispositif physique que votre programme va chercher à contrôler consiste en un véhicule autonome lancé dans une ville virtuelle en deux dimensions. Le but de ce véhicule est de franchir toutes les étapes d'un parcours préétabli tout en respectant un certain nombre de contraintes, parmi lesquelles le respect des limitations de vitesse, des feux rouges, ou encore l'évitement des obstacles. Votre code sera évalué sur sa capacité à arriver au bout de parcours de plus en plus complexes dans le respect des dites contraintes. La qualité de votre code (simplicité, présence de commentaires, modularité, clarté, etc.) sera également évaluée.

Consignes générales. Le projet est à réaliser en binôme. Sa note sera toutefois individuelle, et obtenue via une soutenance finale. Tout **échange de code entre binômes est strictement interdit** et entraînera l'attribution de 0 aux binômes concernés.

2 Environnement de développement

2.1 Avant de débuter

Avant de vous lancer dans la réalisation du projet, vous devez avoir installé :

- un **système d'exploitation de type UNIX** comme GNU/Linux ou macOS,
- la **chaîne de développement C standard** incluant le compilateur GCC et GNU Make 4.0+,
- le **gestionnaire de version Git**,
- la **bibliothèque multimédia SDL2** (C), installable via votre gestionnaire de paquets,

— le **compilateur Heptagon**, installable via OPAM,
— optionnellement, les **bibliothèques Pandas et Matplotlib** (Python), installables via pip.

2.2 Débuter

Votre code, écrit en Heptagon, doit s'insérer dans un squelette composé de code Heptagon et de code C. La distribution de ce squelette, le suivi du projet et le relevé final seront intégralement réalisés via Git. Pour vous lancer dans votre copie du projet, vous devez :

1. forker le dépôt Git du cours,
2. ajouter vos enseignants (*@aguatto* et *@baudart*) à votre fork,
3. passer votre fork en mode privé,
4. éditer le fichier AUTEURS pour y spécifier les membres de votre binôme.

La dernière étape est essentielle : **tout projet dont le dépôt Git est en mode public se verra automatiquement attribué la note de 0**. Une fois ces trois étapes réalisées, vous pouvez commencer à travailler sur le projet en vous aidant des informations disponibles dans les sections suivantes du document.

2.3 Rendre le projet

Le projet est à réaliser avant le

dimanche 18 décembre 2022 à 23h50.

Toute modification ultérieure à cette date sera ignorée. Le rendu se déroulera automatiquement via Git, vous n'avez donc rien de particulier à faire si vous avez suivi les instructions ci-dessus.

Rapport. Votre dépôt doit contenir un rapport présentant succinctement les fonctionnalités réalisées, en insistant sur les éventuels points notables ou originaux de votre solution. Le rapport doit faire **deux pages maximum** et consister en un fichier au **format PDF** présent dans `projet/RAPPORT.pdf` au moment du rendu.

3 Guide du projet

3.1 Architecture du code

Le projet se présente sous la forme d'un ensemble de fichiers écrits en C et en Heptagon, accompagnés d'un `Makefile` et d'un bref README.

Les fichiers écrits en C réalisent diverses tâches utilitaires, ainsi que l'interface graphique du projet. Ils utilisent la bibliothèque SDL2 pour l'interfaçage avec le système. Leur lecture n'est pas obligatoire, mais peut vous éclairer sur le fonctionnement concret du projet.

Les fichiers Heptagon contiennent le code du simulateur, ainsi que le code du contrôleur de véhicule. Nous vous recommandons de les lire avant de commencer à développer votre projet.

— Le fichier `mathext.epi` déclare un jeu de fonctions mathématiques élémentaires.

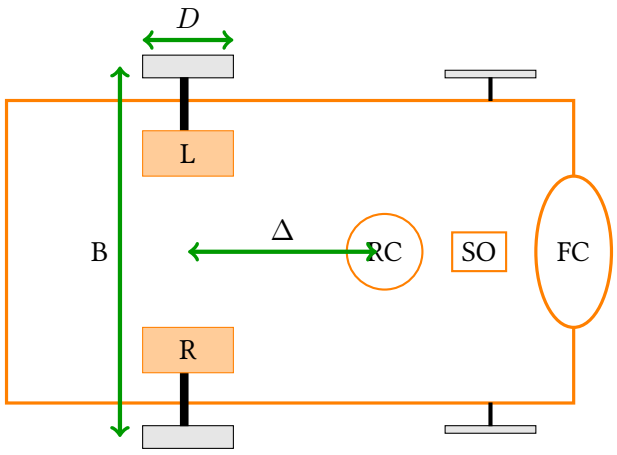

FIGURE 1 – Schéma général du véhicule

- Les fichiers `trace.epi` et `debug.epi` déclarent un jeu de fonctions et nœuds utiles au débogage et à la mise au point de vos programmes. Nous en discuterons en section 5.
- Le fichier `globals.ept` contient les définitions de types et de constantes globales utilisées par le simulateur et votre contrôleur.
- Le fichier `utilities.ept` contient divers nœuds et fonctions utilitaires.
- Le fichier `control.ept` est **le seul fichier que vous devez modifier dans la version finale**, en y implémentant le nœud `controller`. La version qui vous est distribuée contient un contrôleur trivial qui laisse le véhicule inactif.
- Le fichier `vehicle.ept` contient la partie du simulateur chargée du véhicule et de son interfaçage avec le contrôleur.
- Le fichier `city.ept` contient la partie du simulateur chargée de simuler la ville dans laquel le véhicule se déplace.
- Le fichier `challenge.ept` est le fichier Heptagon principal, chargé d'interconnecter les différents composants du simulateur.

Notez donc que **toute modification des fichiers fournis sera ignorée** par l'infrastructure d'évaluation pour la note finale, à l'exception de celles apportées à `control.ept`.

3.2 Fonctionnement du véhicule

Le véhicule que vous devez contrôler est une mini-automobile très simplifiée mais équipée d'actuateurs et de capteurs décrits ci-dessous. La figure 1 en donne une vue schématique.

- Elle dispose de deux roues arrières de rayon D cm. Leurs moyeux sont distants de B cm. Ces roues sont motrices : elles sont connectées par deux axes indépendants à deux moteurs distincts, L (pour *left*) et R (pour *right*). Votre contrôleur **fixe leurs vitesses respectives**.
- Elle dispose de deux capteurs colorimétriques.
  - Le capteur ventral RC (pour *road color*) fournit la couleur de la route sous l'automobile. Il est disposé à une distance Δ cm du point à mi-chemin des moyeux des deux roues.
  - Le capteur frontal FC (pour *front color*) fournit la couleur d'un éventuel feu rouge en

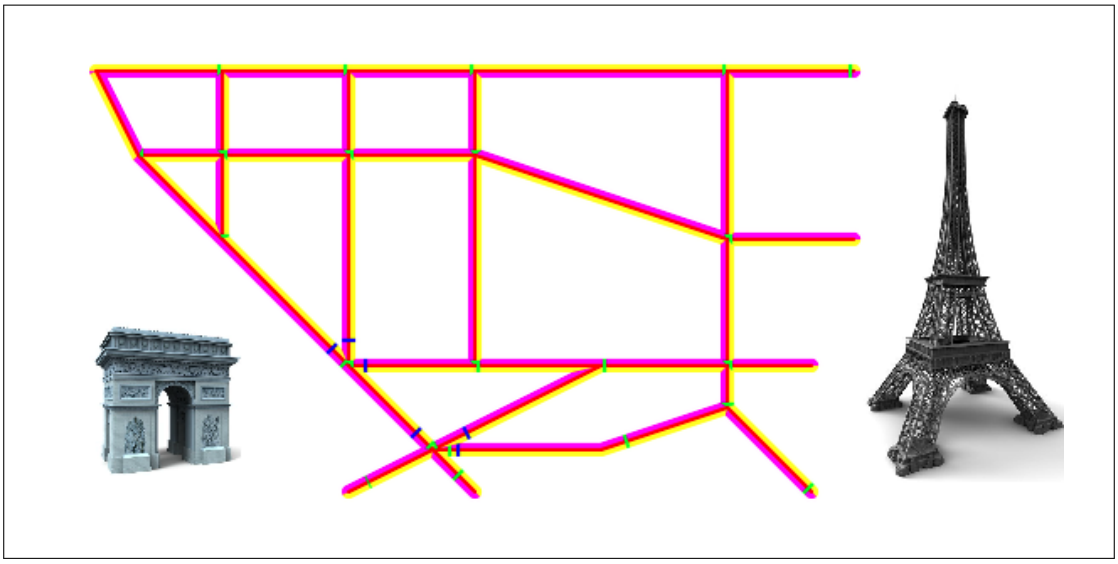

FIGURE 2 – Un exemple de carte

face de l'automobile.
Votre contrôleur peut **lire les couleurs détectées** par ces deux capteurs.

— Elle dispose d'un sonar SO capable de détecter la proximité d'un obstacle (passant). Votre contrôleur a accès à **la distance de l'obstacle** détectée par le sonar.

3.3 Fonctionnement de la ville

Le but de votre code est de contrôler le véhicule afin que celui-ci accomplisse un parcours à travers la ville, sans accident et en respectant l'itinéraire prescrit. Le projet propose une série de villes, les premières étant les plus faciles à traverser. Une des cartes les plus difficiles est représentée à la figure 2.

La route. Chaque ville comprend un certain nombre de routes interconnectées sur lesquelles votre véhicule est censé se déplacer. **Toute sortie de route constitue un accident** qui met fin à la simulation. Les routes sont marquées au sol pour vous aider à éviter un tel sort.

Le marquage est représenté à la figure 3. La bande bleue foncé marque le centre de la route, la bande cyan son côté gauche, la bande magenta son côté droit. Votre véhicule doit chercher à rester au centre, sur la bande bleue. Le contrôleur a accès à cette couleur via son capteur ventral RC. Attention : lorsque votre véhicule se situe à la frontière entre plusieurs bandes, il capture une **combinaison des couleurs de chaque bande**.

En plus des bandes de guidage, le marquage fournit également des bandes vertes et rouges qui signalent respectivement la présence d'une étape ou d'un feu de signalisation. Nous allons décrire ces deux dispositifs et comment ils doivent être pris en compte par votre contrôleur.

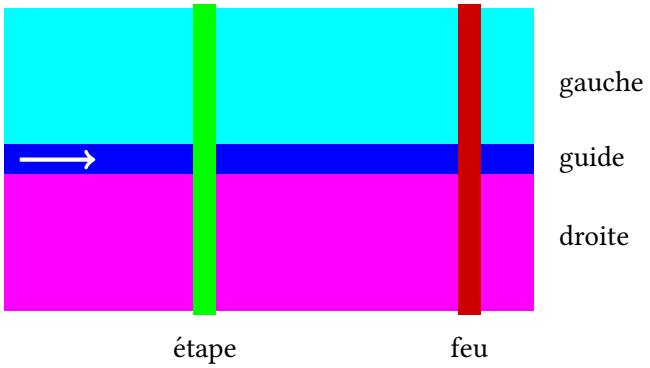

FIGURE 3 – Marquage au sol des routes

L'itinéraire. Chaque ville spécifie un itinéraire auquel votre contrôlleur a accès. Il s'agit d'un tableau d'*actions*. Chaque action appartient à une certaine *catégorie* et spécifie un *paramètre*, ce dernier étant un nombre à virgule flottante dont la signification dépend de la catégorie de l'action. Les trois catégories d'action sont :

1. *Go*, qui indique que le véhicule doit avancer jusqu'à la prochaine étape à la vitesse maximale indiquée par le paramètre ;
2. *Turn*, qui indique que le véhicule doit effectuer une rotation sur lui même, dans le sens inverse des aiguilles d'une montre et d'un angle en degrés indiqué par la valeur du paramètre ;
3. *Stop*, qui indique que l'itinéraire est terminé — la valeur du paramètre n'est pas utilisée.

L'itinéraire est donc principalement formé d'actions *Go* qui indiquent qu'il faut atteindre la prochaine étape, marquée par une bande verte sur la route, et par des actions *Turn*, qui indiquent que le véhicule doit tourner sur lui même à l'étape courante.

Les feux de signalisation. Un marquage au sol de couleur rouge indique la présence d'un feu de signalisation. Si votre véhicule est présent sur un de ces marquages, son capteur frontal RC vous donne accès à la couleur courante du feu. Votre véhicule doit, bien entendu, les **respecter** !

Les obstacles. Un dernier ingrédient est la présence potentielle d'obstacles (passants, etc.) à proximité de la route. Vous devez vous arrêter si vous détectez un obstacle à proximité à l'aide du sonar, faute de provoquer une collision qui serait dommageable à votre véhicule, à l'obstacle, et à votre note finale au projet.

3.4 Compiler et tester son projet

Si vous avez installé les dépendances détaillées en section 2, il suffit d'invoquer `make` pour compiler le projet. Les différentes cartes sont disponibles dans le sous-dossier `assets/`. L'exécutable produit, `scontest`, prend en argument le chemin vers la carte sur laquelle votre contrôleur doit être testé. Par souci de commodité, la cible `make test` lance votre projet sur la première carte, ce qui devrait suffire pendant les premiers temps du développement.

4 Méthodologie suggérée

Il est conseillé d'attaquer ce projet de programmation avec méthode. La quantité de code nécessaire à sa réalisation n'est sans doute pas très importante comparé à d'autres projets que vous avez réalisés pendant vos études, mais ce code peut être assez délicat à concevoir et mettre au point. Après avoir **parcouru le code Heptagon de simulation**, nous vous suggérons de développer votre contrôleur en passant par les étapes fonctionnelles suivantes.

1. Avancer tout droit le long d'une route longiligne, par exemple celle de la carte 00, lorsque le véhicule a été positionné avec un angle correct initialement. Le contrôleur doit être capable d'avancer à la vitesse maximale fournie par l'itinéraire, et ce dans la bonne direction.
2. Corriger un angle initial désaxé, puis suivre une route incurvée — par exemple, celle de la carte 02. (Les notions d'automatique de base fournies en cours peuvent être très utiles ici !)
3. Tourner d'un angle spécifié.
4. Détecter le nouveau segment de route après une rotation.
5. Interpréter les marques d'étape au sol et l'itinéraire.
6. Interpréter les balises d'arrêt indiquant les feux de signalisation, respecter ces derniers.
7. Interpréter le sonar et éviter d'entrer en collision avec les obstacles.

Une fois toutes ces fonctionnalités implémentées, il ne vous reste plus qu'à optimiser la qualité de votre contrôleur sur autant de cartes que possible.

5 Trucs et astuces

Soutien au projet. Toutes les séances de travaux pratiques restantes sont désormais consacrées intégralement à la réalisation du projet. Profitez-en pour discuter avec vos enseignants, qui sont là pour ça, et vos camarades (mais pas d'échange de code !).

Débogage. Le code fourni dispose de fonctionnalités rudimentaires de débogage. Lisez l'interface du module `Debug`, qui permet d'afficher sur la sortie standard les valeurs courantes des flots accompagnées d'un message.

Graphage. Le module `Trace`, permet un débogage un peu plus sophistiqué que celui offert par `Debug`. Il permet de tracer l'évolution de flots booléens, entiers ou flottants au cours du temps — cf. l'interface du module. Son fonctionnement suppose que vous ayez installé les bibliothèques Python que sont Matplotlib et Pandas. Une fois ceci fait, pour afficher les courbes, vous devez lancer votre binaire `scontest` avec l'outil `hept-plot` fourni dans le dossier `tools/` du dépôt du cours.

6 Versions de ce document

Ce projet est l'adaptation à Heptagon par A. Guatto de celui co-réalisé en SCADE par E. Asarin et M. Sighireanu de 2017 à 2019.

02/11/2022 Version initiale.

08/11/2022 Correction date de rendu et typos.