Projet long M1
Robot suiveur de ligne

Yani Amrioui Idir Lankri

30 mai 2017

Introduction

Problème

Programmer un robot capable d'aller d'un point de départ à un point d'arrivée en suivant une ligne

Exemples d'application

- Manutention : déplacer des charges lourdes, des marchandises dans un entrepôt ou une usine
- Aider des personnes à mobilité réduite dans leur vie de tous les jours (ex : apporter des objets)

Fonctionnalités

- Apprentissage de couleurs
- Reconnaissance de couleurs
- Suivi de lignes droites et courbes

Scénario typique d'utilisation

Contexte

On dispose d'un circuit (ligne tracée sur un certain fond) et on veut aller d'un point A à un point B de ce circuit en suivant ce circuit.

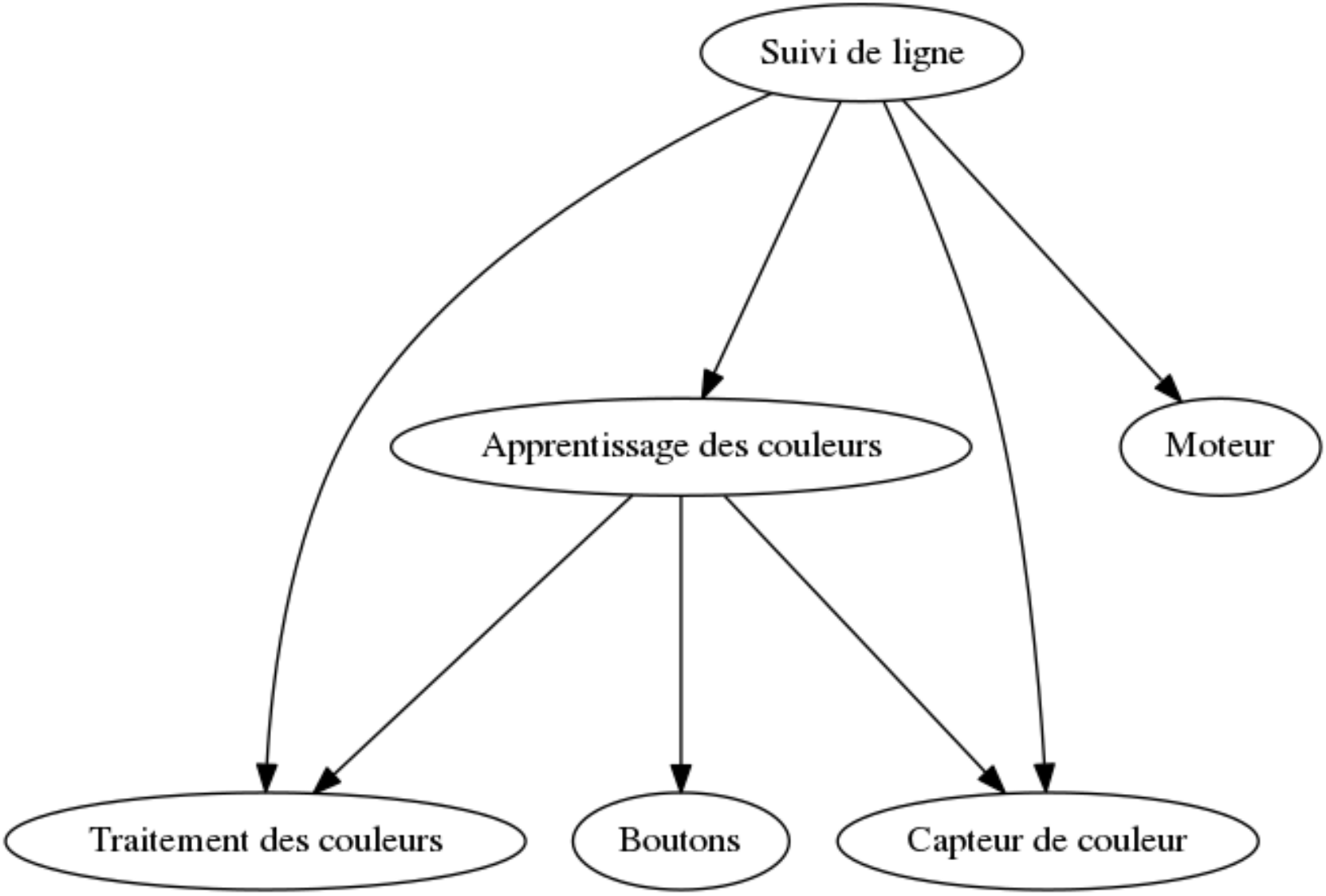
2 étapes

1. Apprendre au robot les couleurs du circuit
2. Placer le robot sur le circuit, puis le lancer afin qu'il suive la ligne

Décomposition du problème

1. Faire apprendre les couleurs au robot
2. Faire que le robot soit capable de reconnaître ces couleurs
3. Suivre une ligne en traitant les données obtenues via le capteur de couleur

Architecture du logiciel



Points techniques

Interaction avec le robot

Communication avec le robot via le système de fichiers (*device files*)

Compilation croisée

- Problème : générer du code machine exécutable par le robot sur une autre architecture
- Solution : Docker

Programmation

```
(* [member col col_cloud] returns true if and only if [col]
    belongs to [col_cloud].  *)
let member col col_cloud =
  (* We suppose that colors composing a color cloud follows a Gaussian
     distribution.  Thus, a random color of the cloud is between
     center - 3*sigma and center + 3*sigma with probability 99.7%.  *)
  let delta = shift 3. col_cloud.sigma in
  geq col (diff col_cloud.center delta) &&
  leq col (add col_cloud.center delta)

(* Return the cloud [c] among [col_clouds] such that the distance
   between [c.center] and [col] is minimal.  *)
let nearest_cloud col col_clouds =
  let aux col (old_cloud, old_dist) new_cloud =
    let new_dist = dist col (center new_cloud) in
    if new_dist < old_dist then (new_cloud, new_dist) else (old_cloud, old_dist)
  in
  match col_clouds with
  | [] -> assert false
  | col_cloud :: col_clouds ->
    let dist = dist col (center col_cloud) in
    fst (List.fold_left (aux col) (col_cloud, dist) col_clouds)

let recognize col known_cols =
  match List.filter (member col) known_cols with
  (* There is no color recognized without doubt.  In this case we return
     the "nearest" color among known colors.  *)
  | [] -> Probable.Maybe (nearest_cloud col known_cols)

  (* We decide between the candidates by computing the "nearest"
     cloud.  *)
  | col_clouds -> Probable.Sure (nearest_cloud col col_clouds)
```

Conclusion

Enseignements de ce projet

- Découverte de la robotique et de la programmation de système embarqué
- Difficulté de développer et tester un logiciel dépendant de capteurs

Améliorations possibles

- Utiliser un historique de navigation
- Gérer des circuits complexes : sélection de chemin par un code couleur