Logique floue
`Sauvons le RMS Titanic`

Guillaume Laroyenne

28 septembre 2021

Résumé

Ce document présente une synthèse du projet de logique floue que nous avons réalisé en deuxième année. Le cœur de ce projet est une bibliothèque de logique floue que nous avons programmée en *C++*. La bibliothèque comporte les principaux opérateurs de bases de la logique floue ainsi que quelques méthodes de défuzzification. Pour que les utilisateurs puissent manipuler plus simplement cette bibliothèque, nous avons décidé d'ajouter un interpréteur permettant de modéliser des problèmes de logique floue sans avoir à la manipuler directement. La logique floue permettant de prendre des décisions complexe à partir d'un ensemble de données. Nous avons décidé de l'utiliser afin de réaliser un pilote automatique évitant les obstacles pouvant se trouver devant un navire, tout en utilisant la bibliothèque que nous avons conçu.

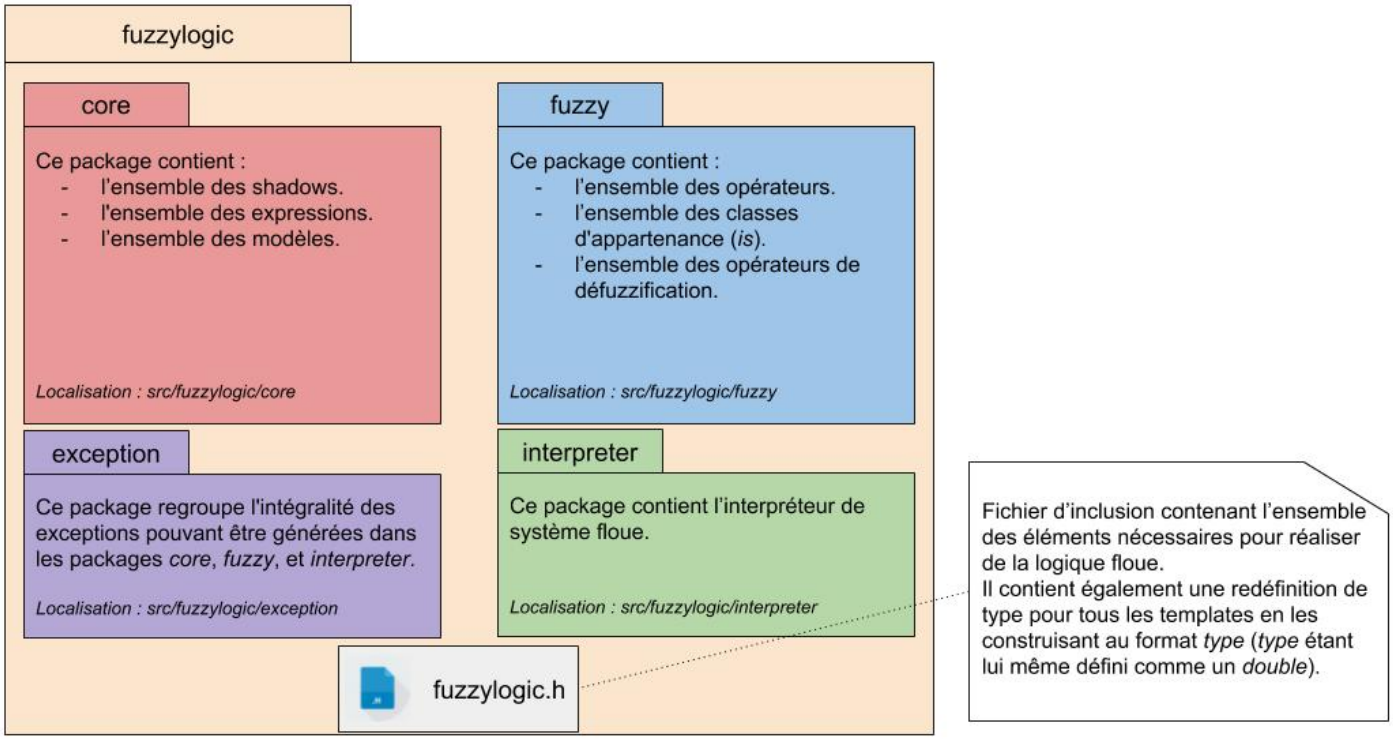
1 Construction de la bibliothèque de logique floue

Pour toute information sur l'arborescence, la compilation et l'utilisation de la bibliothèque, veuillez vous référer au fichier **README.md** consultable à la racine du projet à l'adresse :
`https://github.com/LaroyenneG/fuzzy-logic`

Notre *framework* a été développé en langage *C++*, compilé avec le compilateur GNU *g++* et *CMake*. Afin de s'assurer du bon fonctionnement de la bibliothèque, nous avons utilisé le *framework CPPUNIT* permettant de réaliser des tests unitaires en langage *C++*. Toutes les opérandes et opérateurs sont vérifiés via un filet de test permettant d'identifier d'éventuels mauvais fonctionnements. Cela permet d'effectuer des modifications sur la bibliothèque et d'identifier rapidement les possibles régressions.

1.1 Organisation logiciel du projet

FIGURE 1 – UML des packages du *framework* de logique floue



1.2 Contenu du projet

TABLE 1 – Liste des opérateurs et opérandes du *framework* de logique floue

Nom	Type	Description
AggPlus	Agg	Retourne l'addition des évaluations des valeurs d'entrées entre 0 et 1
AggMax	Agg	Retourne la valeur maximale des évaluations des valeurs d'entrées
AndMin	And	Retourne la valeur minimale des évaluations des valeurs d'entrées
AndMult	And	Retourne la multiplication des évaluations des valeurs d'entrées
ApproximatelyEqual	Or	Retourne la valeur de l'évaluation si les évaluations des valeurs d'entrées sont égales à 0,2 près sinon retourne 0
Equal	Or	Retourne la valeur de l'évaluation si les évaluations des valeurs d'entrées sont égales sinon renvoie 0
NotMinus1	Not	Retourne la valeur de l'évaluation la résultat de 1 moins l'évaluation de l'opérateur
OrMax	Or	Retourne la valeur maximale des évaluations des valeurs d'entrées
OrPlus	Or	Retourne l'addition des évaluations des valeurs d'entrées entre 0 et 1
ThenMin	Then	Retourne la valeur minimale des évaluations des valeurs d'entrées
ThenMult	Then	Retourne la multiplication des évaluations des valeurs d'entrées
IsBell	Is	Retourne la valeur de l'évaluation de l'opérande suivant une cloche
IsBornedTrapezoid	Is	Retourne la valeur de l'évaluation de l'opérande suivant un trapèze bornée en abscisse
IsGaussian	Is	Retourne la valeur de l'évaluation de l'opérande suivant une courbe de Gauss
IsParable	Is	Retourne la valeur de l'évaluation de l'opérande suivant une parabole
IsRampLeft	Is	Retourne la valeur de l'évaluation de l'opérande suivant une rampe commencent par la gauche
IsRampRight	Is	Retourne la valeur de l'évaluation de l'opérande suivant une rampe commencent par la droite
IsRangeBell	Is	Retourne la valeur de l'évaluation de l'opérande suivant une cloche bornée en abscisse
IsSingleton	Is	Retourne la valeur de l'évaluation de l'opérande suivant une valeur unique
IsSigmoid	Is	Retourne la valeur de l'évaluation de l'opérande suivant une sigmoïde
IsTrapezoid	Is	Retourne la valeur de l'évaluation de l'opérande suivant un trapèze
IsTriangle	Is	Retourne la valeur de l'évaluation de l'opérande suivant un triangle

2 Mise en place d'un interpréteur de logique floue

Pour réaliser des systèmes floue plus simplement, nous avons réalisé un interpréteur utilisant notre bibliothèque. Il réalise les manipulations sur la *factory* en fonction des instructions d'un texte d'entrée. Cela permet de se rapprocher du langage humain, afin de se concentrer uniquement sur la logique et non de l'aspect programmation.

FIGURE 2 – Schéma présentant le concept de l'interpreteur

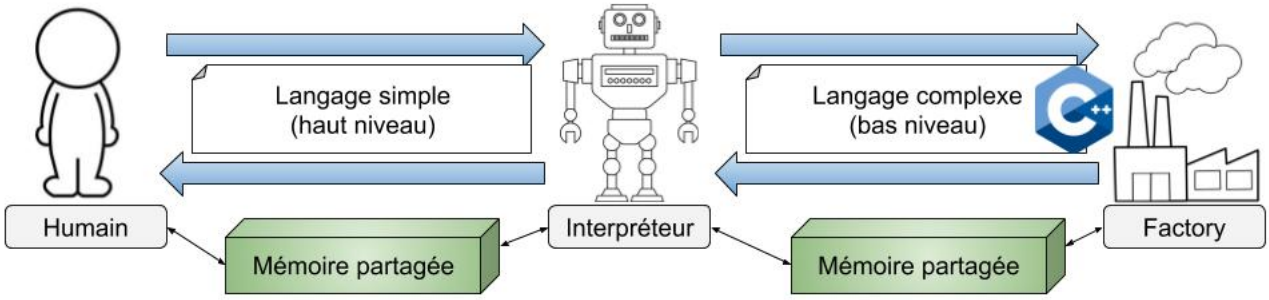


FIGURE 3 – Exemple de code de l'interpréteur pour l'évaluation d'un pourboire

```
FuzzySystem Tip Cog NotMin AndMin OrMax ThenMin AggMax

# Variable zone
Var Input Tip->service
Var Input Tip->food
Var Output Tip->tip

# Shapes definition
Define Triangle Tip->cheap 0.0 5.0 10.0
Define Triangle Tip->average 10.0 15.0 20.0
Define Triangle Tip->generous 20.0 25.0 30.0
Define RampRight Tip->rancid 0.0 5.0 5.0
Define RampLeft Tip->delicious 5.0 10.0 10.0
Define RangeBell Tip->poor 1.0 1.0 0.0 0.0 5.0
Define Bell Tip->good 1.2 3.2 5.0
Define RangeBell Tip->excellent 1.0 1.0 10.0 5.0 10.0

# Define rules
Rules Tip {
If (((service Is poor) Or (food Is rancid)) Then (tip Is cheap))
If ((service Is good) Then (tip Is average))
If (((service Is excellent) Or (food Is delicious)) Then (tip Is generous))
}

# Build rules and system
Build Tip 0 25 1
```

3 Création du simulateur naval

3.1 Analyse des phénomène physique et mécanique

Pour réaliser notre simulateur naval, nous avons du étudier les différents phénomènes physiques qui s'exercent sur un navire ainsi que ces différents composants techniques.

TABLE 2 – Caractéristiques techniques du Titanic saisies dans le simulateur

Longueur	269 m	Puissance d'une machine alternative	15000 ch
Maître-bau	28 m	Nombre de pale des hélices latérales	3
Tirant d'eau	10,5 m	Diamètre des hélices latérales	7,2 m
Vitesse Maximal	23 / 24 nœuds	Diamètre de l'hélice centrale	5,20 m
Poids	46000 tonnes	Surface du gouvernail (immergé)	42 m^2
Puissance de la turbine	16000 ch	Nombre de pale de l'hélice centrale	4

Tel un objet physique, dans le simulateur le Titanic possède une position, une vitesse, une orientation et une accélération. Pour qu'il puisse se déplacer, il faut donc déterminer l'accélération. Selon Newton : $\sum \overrightarrow{F_i} = m \times \overrightarrow{a}$, il faut donc déterminer toutes les forces qui s'exercent sur le navire pour connaître l'accélération.

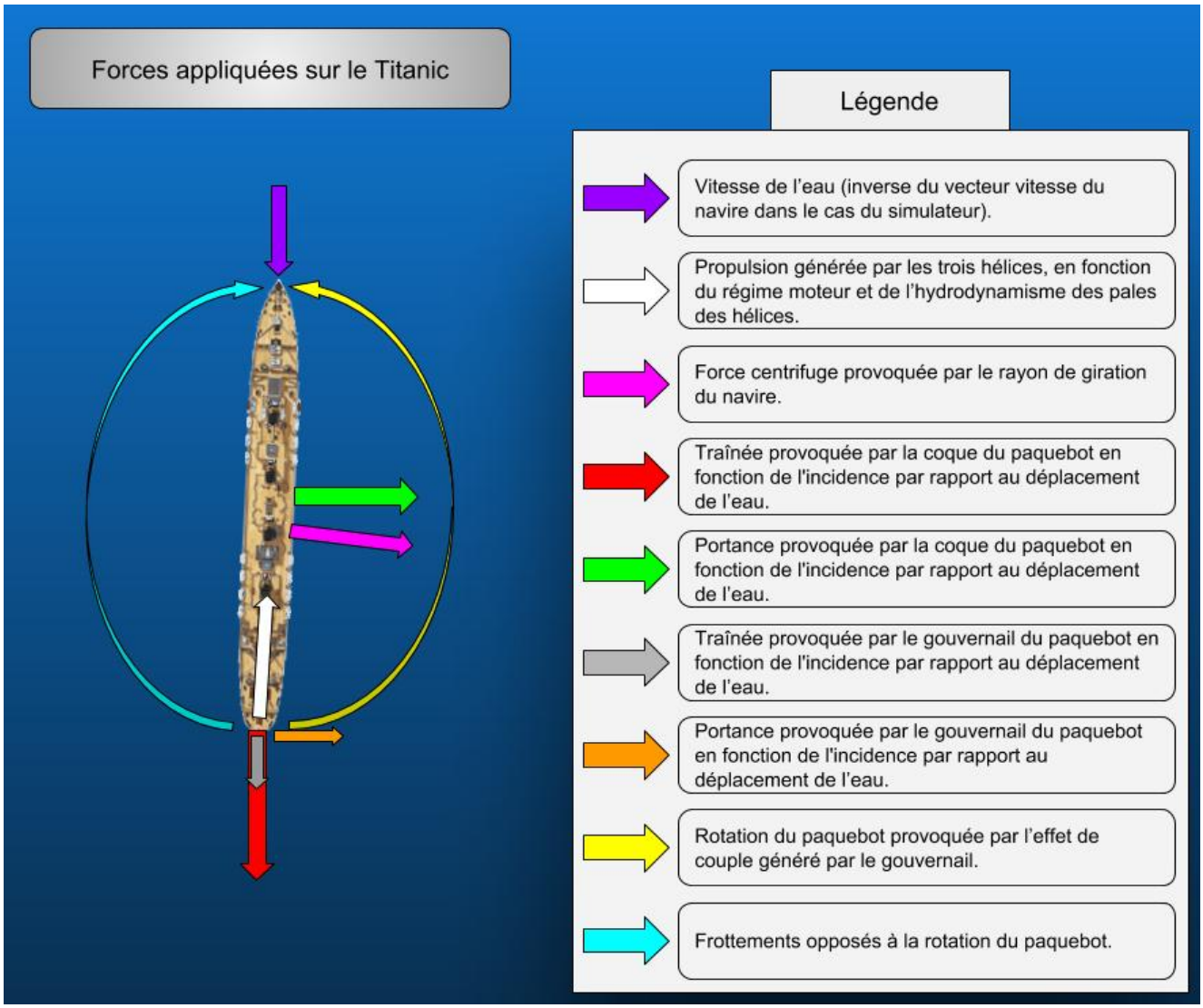
La première force à identifier est la force de propulsion générée par les hélices en raison du principe d'action réaction. Elle dépend de son profil physique (diamètre, incidence des pales, surface...) et de sa vitesse de rotation.
Lorsque le navire tourne il subit également une force centrifuge, qui peut être calculée en identifiant le centre du cercle décrit par la trajectoire. Selon les lois de la mécanique des fluides, lorsque le navire se déplace dans l'eau, sa coque et son gouvernail subissent des forces appelées portances et traînée. La combinaison de la portance et de la traînée produit une résultante hydrodynamique, qui est la force anti-dérive du navire. La portance et la traînée sont calculées à l'aide des formules suivantes :

$$F_z = \frac{1}{2} \times \rho \times S \times V^2 \times C_z$$
$$F_x = \frac{1}{2} \times \rho \times S \times V^2 \times C_x$$
$$\overrightarrow{R_h} = \overrightarrow{F_z} + \overrightarrow{F_x}$$

Avec :
- F_z la portance en N (perpendiculaire à la vitesse du fluide).
- F_x la traînée en N (opposée au vecteur vitesse).
- ρ la masse volumique du fluide en kg/m^3.
- S la surface de référence en m^2.
- V la vitesse de déplacement du fluide en m/s.
- C_z coefficient de portance en fonction de l'incidence (sans unité).
- C_x coefficient de traînée en fonction de l'incidence (sans unité).
- $\overrightarrow{R_h}$ résultante hydrodynamique.

La rotation du navire est provoquée par la résultante hydrodynamique du gouvernail situé à l'arrière, induisant un effet de couple.
À cette rotation s'opposeront des forces de frottements, en raison des frottements avec l'eau sur la coque. Pour le calcul de cette force nous avons choisi un coefficient de traînée de 1,1.

FIGURE 4 – Schéma présentant les forces physiques appliquées sur le Titanic dans le simulateur



On comprend donc que piloter précisément un navire de cette taille n'est pas simple. Avec toutes les forces physique qui s'appliquent sur lui, il fraudais réaliser beaucoup de calculs pour déterminer les bonnes manœuvres à réaliser. C'est pour cela que nous avons avons choisi d'utiliser la logique floue pour réaliser un pilote automatique ; car elle permet d'abstraire tous les phénomènes physique.

3.2 Programmation de l'interface

Le simulateur naval a été réalisé en langage *C++* en respectant le *design pattern Model View Controller*. L'interface graphique du simulateur a été développé à l'aide du *framework Qt5*.

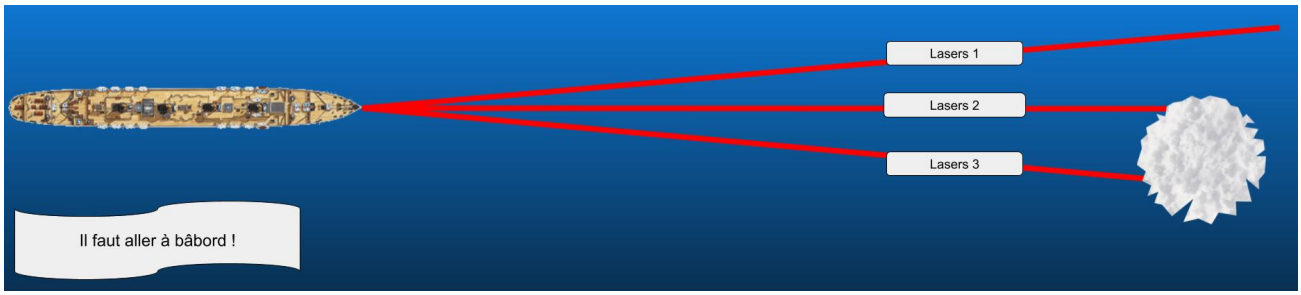
- La section **modèle** contient l'ensemble éléments pour réaliser la gestion des phénomènes physique et mécaniques.
- La section **vue** comportent le système de virtualisation (transforment le modèle en scène 2D), ainsi que les composants de l'interface.
- La section **contrôleur** contient tous les actions à réaliser pour chaque composant graphique, ainsi que le pilote automatique.

4 Création d'un pilote automatique pour le paquebot

4.1 Objectifs et besoins

Le pilote automatique doit permettre au Titanic d'éviter les icebergs en toute sécurité avant même que l'équipage puisse les voir. Mais pour éviter les icebergs, il est nécessaire de les détecter. Pour cela nous avons ajouté à la proue trois capteurs lasers indiquant une distance en pourcentage par rapport à sa portée. Si un capteur indique 100% alors il n'y a pas d'objet. En revanche si un capteur indique 50% alors il y a un objet à la moitié de la potée maximal du capteur. Les trois lasers ont été placé avec un petit angle, afin de pouvoir déterminé le meilleur côté par le quel l'objet peut être évité.

FIGURE 5 – Schéma présentant les capteurs lasers du Titanic



Pour déterminer la meilleur portée des lasers, il faut connaître la capacité à virer de bord du navire. Dans les meilleurs conditions, le Titanic avait un rayon de giration de 750 m. Nous avons donc réglé la portée des lasers à 800 m, ce qui devrait permettre d'anticiper l'approche des objets suffisamment tôt.

4.2 Modélisation du pilote automatique en logique floue

Le but : Agir sur l'angle de la barre pour changer (maintenir) le cap pour éviter que le paquebot s'écrase sur l'iceberg.
Le résultat de ce système est un pourcentage. Il correspond à l'angle de la barre pour laquelle le paquebot ne heurtera pas l'iceberg.

- Lorsque la barre aligne le gouvernail avec l'axe du bateau, elle est a une valeur de 50%.
- Pour une manœuvre à bâbord ("gauche"), il faut que la barre se situe dans la zone 0% - 49%.
- Pour une manœuvre à tribord ("droite"), il faut que la barre se situe dans la zone 51% - 100%.

FIGURE 6 – Explication de la relation entre la barre et le système flou

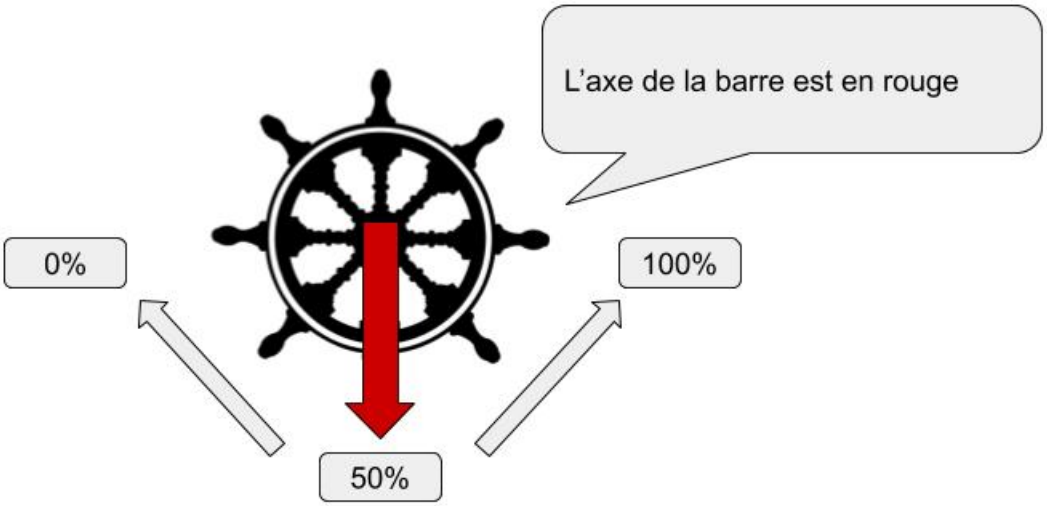


FIGURE 7 – Explication du système floue du pilote automatique

4.3 Réalisation du système floue à l'aide du *framework*

Une fois les concepts théoriques saisis, nous avons exprimé notre pilote automatique dans le langage de l'interpréteur.

FIGURE 8 – Code du pilote automatique pour l'interpréteur

```
FuzzySystem Helm Cog NotMin AndMin AEqual ThenMin AggMax

# Variables definition
Var Input Helm->laser1
Var Input Helm->laser2
Var Input Helm->laser3
Var Output Helm->value

# Shape definition
Define Triangle Helm->laser 0.0 0.0 1.0
Define Trapezoid Helm->front 0.98 1.0 0.98 1.0
Define Triangle Helm->left 0 0.25 0.5
Define Trapezoid Helm->central 0 1 0 1
Define Triangle Helm->rightTri 0.5 0.75 1.0
Define Trapezoid Helm->rightTra 0.5 1 0.5 1.0

# Rules bloc
Rules Helm {
If ((laser1 Is laser) Then (value Is rightTri))
If ((laser3 Is laser) Then (value Is left))
If (((laser1 Is laser) Or (laser3 Is laser)) Then (value Is rightTra))
If ((((laser1 Is front) And (laser2 Is front)) And (laser3 Is front)) Then (value
}

# System building
Build Helm 0.0 1.0 0.001
```