Manuel d'utilisation

Rover

Fait par :
M'barek ZACRI

2020

Table des matières

1 Architecture du robot

1.1 Architecture matérielle

La figure suivante présente l'intégration des différents composants de l'architecture du rover :

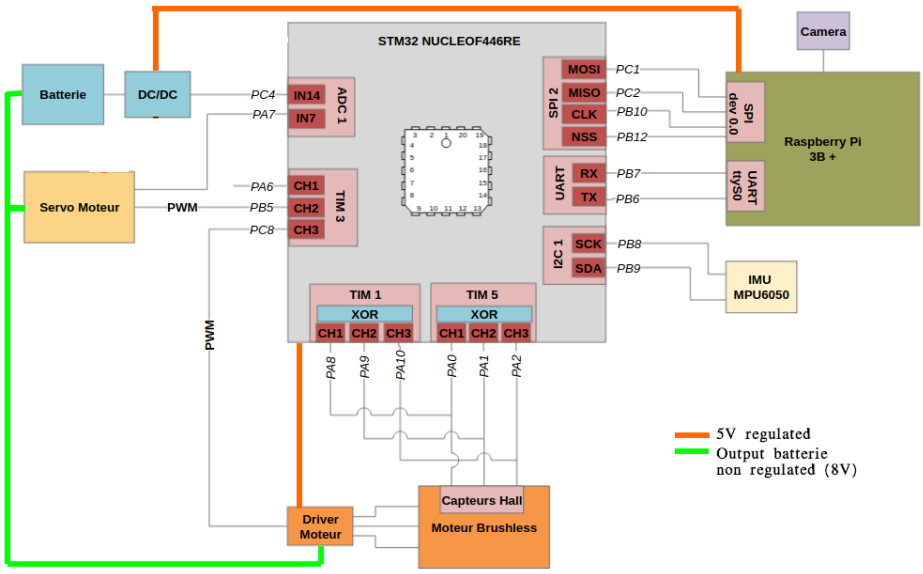


FIGURE 1 – Architecture matérielle

Remarque : La communication UART entre la Raspberry et le micro-contrôleur est supprimée au fur et à mesure du développement. Néanmoins, l'historique git permet de la restaurer pour toute fin de débogage. La figure suivante présente la connectique du micro-contrôleur STM32F446RE utilisé :

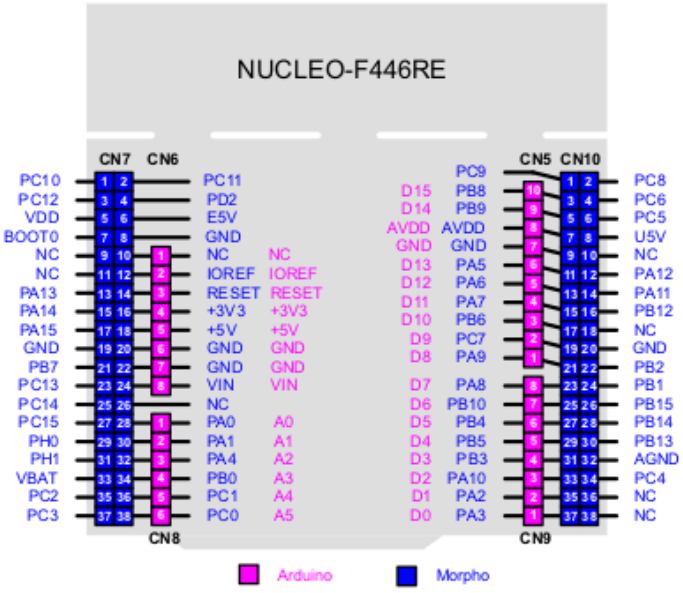


FIGURE 2 – Nucleo STM32F446RE

1.2 Architecture logicielle

1.2.1 Rôles et séparation des tâches :

Les rôles des composants sont séparés comme-ci :

- Raspberry Pi : Exécuter l'environement ROS, envoyer la commande au micro-contrôleur et recevoir l'état du véhicule (odométrie, batterie...)
- Micro-contrôleur : Exécuter la commande de la Raspberry et renvoyer l'état. Il interagit donc avec les actionneurs et les capteurs embarqués (sauf la caméra).

1.2.2 Architecture du code micro-contrôleur

Structure du code :

```
├── Inc
│   ├── Communication
│   │   └── Raspberry.h
│   ├── Control
│   │   └── Control.h
│   ├── FreeRTOSConfig.h
│   ├── main.h
│   ├── Peripherals
│   │   ├── ADCConfig.h
│   │   ├── DMAConfig.h
│   │   └── timersConfig.h
│   ├── RoverConfig.h
│   ├── Sensors
│   │   ├── AckermannOdom.h
│   │   ├── Filters
│   │   │   ├── BrushlessSpeed_Kalman.h
│   │   │   ├── EulerOrientationCompFilter.h
│   │   │   ├── LowPassFilters.h
│   │   │   └── ModelDrivenFilters.h
│   │   ├── MPU6050
│   │   │   └── sd_hal_mpu6050.h
│   │   └── RoverStateSensors.h
│   ├── stm32f4xx_hal_conf.h
│   └── stm32f4xx_it.h
├── SLAM_Rover.doxyfile
├── Src
│   ├── Communication
│   │   └── Raspberry.c
│   ├── Control
│   │   └── Control.c
│   ├── freertos.c
│   ├── main.c
│   ├── Peripherals
│   │   ├── ADCConfig.c
│   │   ├── DMAConfig.c
│   │   └── TimersConfig.c
```

- Sensors
  - AckermannOdom.c
  - Filters
    - BrushlessSpeed_Kalman.c
    - EulerOrientationCompFilter.c
    - LowPassFilters.c
    - ModelDrivenFilters.c
  - MPU6050
    - sd_hal_mpu6050.c
  - RoverStateSensors.c
- stm32f4xx_hal_msp.c
- stm32f4xx_hal_timebase_tim.c
- stm32f4xx_it.c
- syscalls.c
- sysmem.c
- system_stm32f4xx.c

Les tâches temps réel : Deux tâches temps-réels sont ordonnancés par FREERTOS. La tâche principale s'intitule "ControlTask". La figure suivante présente la structure de la tâche :

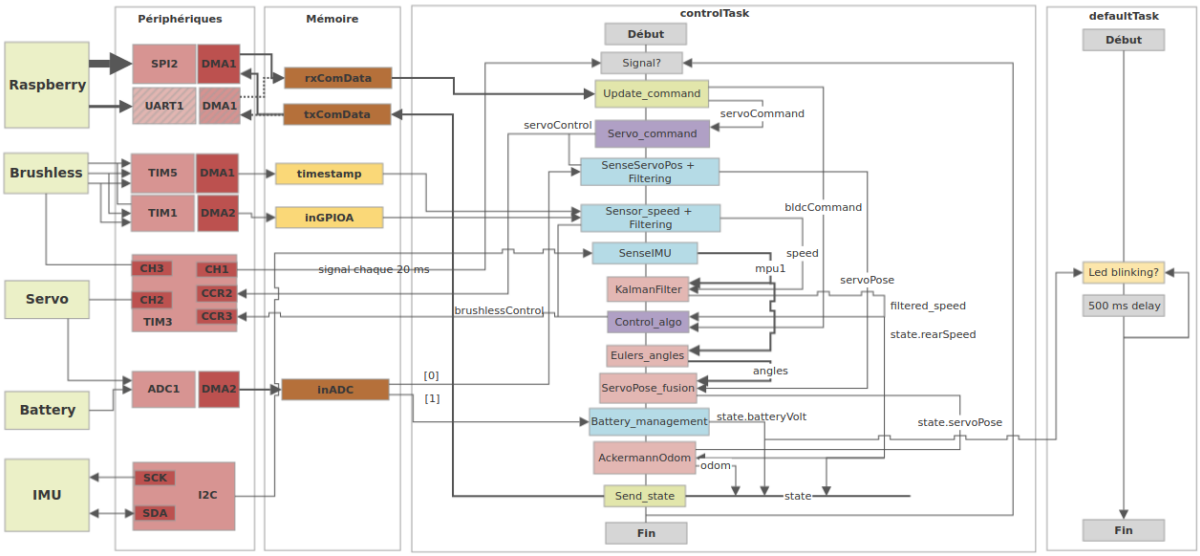


FIGURE 3 – Architecture logicielle du micro-contrôleur

Remarque : Quelques mise à jour du code sont faites après le tracé de cette figure. Néanmoins, elle permet de visualiser la structure de la tâche.

Visualisation des tâches à l'aide de SEGGER SystemView La librairie SystemView de SEGGER est ajouté au code. Elle permet d'enregistrer l'ensemble des évènements de l'ordonnancement des tâches afin de tracer , à l'aide d'une interface de chez SEGGER qu'on installe sur l'ordinateur, le chronogramme de l'exécution des tâches :

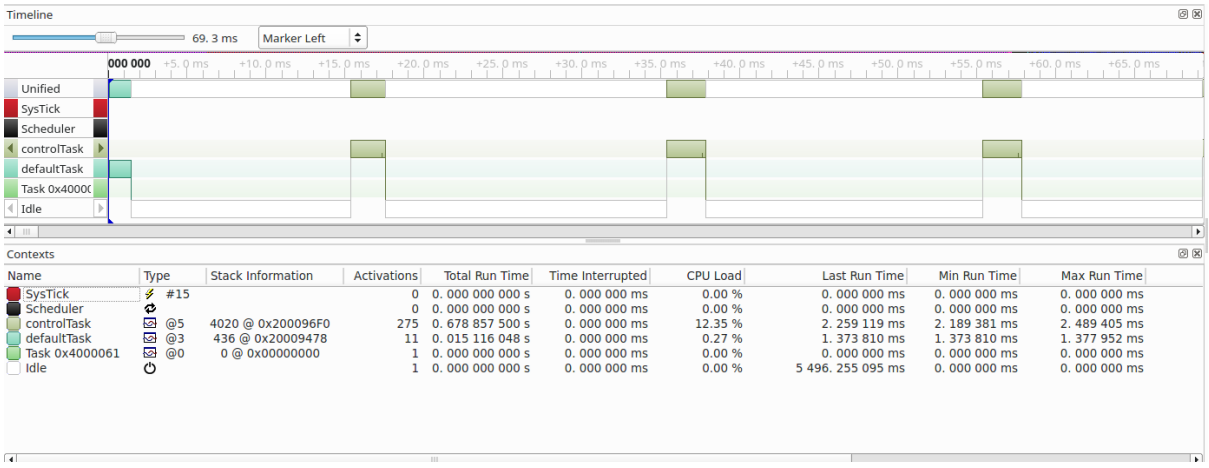


FIGURE 4 – SystemView chronogramme

Pour utiliser SystemView avec FREERTOS 10.0.1 sous CMSIS 1.02, les étapes suivantes sont à suivre :

1. Après chaque génération de code à partir du fichier .ioc, appliquez le patch décrit dans le fichier Core/SystemView/portsPatch.txt. Les étapes sont à l'intérieur.
2. Décommenter la définition de la variable préprocesseur SYSTEMVIEW dans le fichier de configuration RoverConfig.h (ligne 52).
3. En cours de l'exécution du code, une zone mémoire , dont la taille est configurable, enregistre les évènements. En utilisant l'onglet "Live Expressions" on peut suivre le remplissage de cette zone. La structure concernée est la suivante : _SEGGER_RTT.aUp[1]. À l'intérieur de cette structure, quand la variable WrOff atteint SizeOfBuffer, l'enregistrement est terminé.
4. Une fois remplie, il suffit d'exporter la zone mémoire en fichier binaire afin d'être utilisé par l'interface SystemView qu'on a installé sur l'ordinateur. Pour ce faire, on utilise l'onglet memory de CubeIDE. Il permet de "monitorer" une zone mémoire et donc la zone _SEGGER_RTT.aUp[1].pBuffer qui nous intéresse. Il faut donc l'ajouter puis l'exporter en spécifiant la taille de la zone à exporter (n'oubliez pas l'extension .bin lors de l'enregistrement).
5. À l'aide du fichier X.bin enregistré, on peut visualiser l'ensemble des informations en chargeant ce fichier dans l'interface SystemView installée.

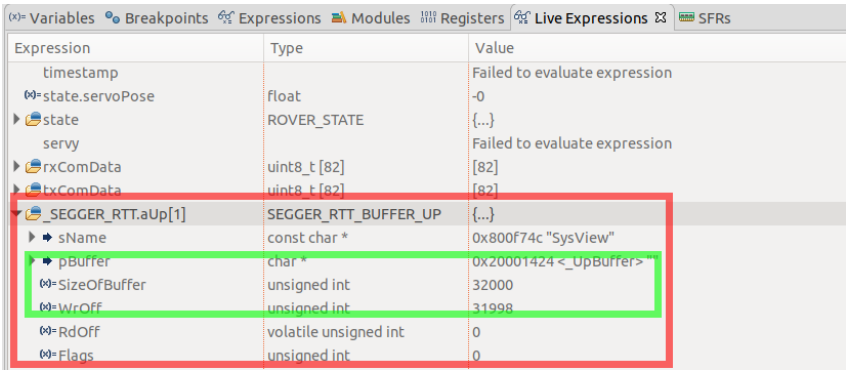


FIGURE 5 – Capture Live Expressions

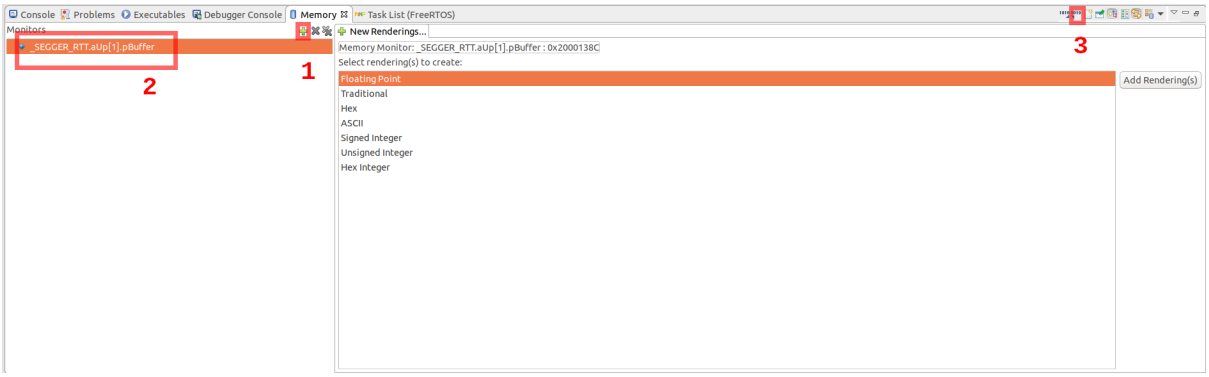


FIGURE 6 – Capture onglet Memory

1.2.3 Architecture du code de la Raspberry Pi

La visualisation rqt_graph permet de spécifier l'architecture des noeuds lancés par le launchfile ROS principal (main.launch du package rover_supervisor) :

FIGURE 7 – Figure rqt_graph

Les noeuds externes sont colorés en vert et les noeuds propres au rover en rouge. Le noeud rover_node permet de jouer le rôle d'interface entre la Raspberry et le micro-contrôleur :

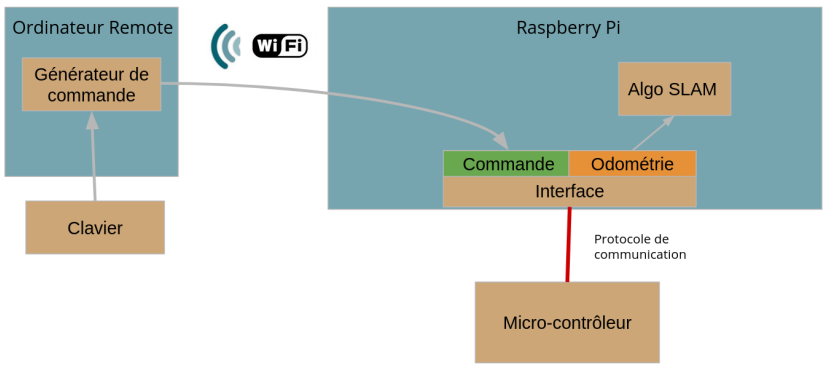
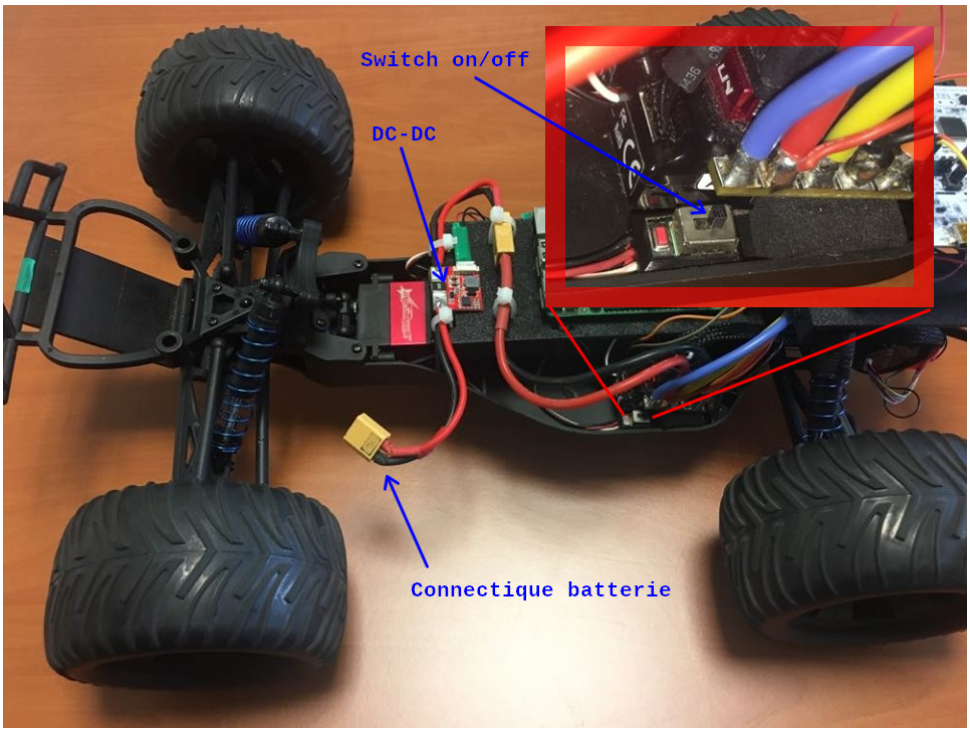


FIGURE 8 – Figure rqt_graph

2 Mise en route

2.1 Allumage du rover

Pour allumer la Raspberry Pi, il suffit de connecter la batterie :



Une fois la Raspberry Pi est en marche, elle émet un point d'accès qui s'intitule "RoverTheBeast". Pour se connecter sur la raspberry, il faut suivre les étapes suivantes :

1. Entrer le mot de passe du point d'accès. Mdp :Rover123
2. Se connecter à travers ssh à l'utilisateur rover dont l'adresse IP est 172.10.10.1. Commande : ssh rover@172.10.10.1

Un fois connecté à la Raspberry, on peut lancer le launchfile main.launch du package rover_supervisor (commande : roslaunch rover_supervisor main.launch). Veuillez vous refferez à l'ensemble des README qui accompagnent les packages supplémentaires (i.e odom_slam, bag_launcher, tags_marker, ekf_slam ...).

L'interrupteur du rover doit être allumé juste avant le launchfile. Il permet d'allumer les moteurs et le micro-contrôleur. Ensuite, il faut attendre que la led jaune du micro-contrôleur se mette à clignoter avant d'nevoyer une commande. Si ce n'est pas le cas la phase d'initialisation de la centrale inertielle s'est mal passée, veuillez atteindre l'interrupteur et le ré-allumer. Si la led s'etteint en-cours d'utilisation et le rover s'arrête, c'est la tension de la batterie qui descends au dessous du seuil spécifié dans le fichier de configuration (Cf plus Configuration et re-programmation). Ce processus est à répéter à chaque essai.

2.2 Commande

Pour commander le rover, il suffit de lancer un noeud ROS qui permet de publier la commande sur le topic cmd_vel du noeud rover_node.

3 Configuration et reprogrammation

3.1 Fichier de configuration

Pour configurer le code du micro-contrôleur, le fichier RoverConfig.h (Cf architecture logicielle) est disponible. Il permet de configurer les paramètres suivantes :

1. CONTROL : Soit OL (pour Boucle ouverte) ou RE (pour retour d'état). Pour des soucis de stabilité, OL est par défaut.
2. COMMAND_SOURCE : Soit RASP pour recevoir la commande de la part de la Raspberry ou INT (pour internal) si on veut conduire des tests en générant une commande qui oscille entre la valeur COMMAND (paramètre suivant) au sens positive et cette même valeur au sens négatif. La tâche secondaire "defaultTask" permet de définir cette commande à un rythme donné.
3. MAX_COMMAND : Si on reçoit une commande supérieure à cette valeur ou la vitesse mesurée est supérieur, le rover s'arrête.
4. SENSOR : choix du capteur vitesse, par défault sur 4.
5. MINSPEED : Permet de calculer la valeur de ticks à partir duquelle on considère que le motor ne tourne pas. Donc la valeur de vitesse détectable au minimum.
6. MAXSPEED : De même
7. MIN_POSITIVE_COMMAND : Pour guarantir une stabilité autour des petites valeurs de vitesse, une valeur minimale de commande dans le sens positif doit être spécifié sachant l'étape d'identification du moteur.
8. MIN_NEGATIVE_COMMAND : De même
9. DEAD_ZONE : Utilser ou non les seuils de commande pour la boucle fermée.
10. PUNCH : la valeur de coup de commande à donner au démarrage pour éviter les décrochages du moteur.
11. CNT_PUNCH : nombre de ticks de 20 ms pendant lesquelles il faut tenir la commande PUNCH.
12. SERVO_ZERO_POSE : La commande donnée au servo pour le mettre sur la position centrale. Cette commande est ajustée sachant la position précédente des roues, car il faut donner plus de commande dans le sens inverse à la position précedente pour atteindre la position centrale (Cf paramètre ADJUST).
13. ESTIMATE_SERVO_POSE : Considérer ou non la sortie du modèle de servo comme le retour du capteur d'orientation des roues. C'est mis à 1 par défault.

3.2 Paramétrage et réglage des capteurs-filtres :

Pour "tuner" les capteurs et les filtres du rover, la discrétisation d'un premier-ordre est essentielle à connaître. L'équation récurrente d'un premier ordre dont les paramètres sont K pour le gain et τ pour le temps de réponse est donnée par :

$$sortie_k = (b * (commande_k + commande_{k-1}) - sortie_{k-1})/a \quad (1)$$

Avec $a = (T_e + 2\tau)/(Te - 2\tau)$, $b = K * T_e/(T_e - 2\tau)$ et T_e la période d'échantillonage.

Modèle du servo moteur Le servo est modélisé par un premier ordre. Le gain est étalonné pour avoir les valeurs maximales des orientations pour une commande maximale. Le temps de réponse est choisi à 60 ms. Pour paramétrer ce modèle le gain pour être modifié sachant la qualité de l'odométrie quand le rover tourne.

Modèle du moteur sans balais et filtrage Le modèle du moteur sans balais permet de filtrer le bruit due au décrochage du moteur. Pour paramétrer ce filtre, on peut changer le modèle sachant l'étape d'identification. Il est aussi possible de modifier la largeur de la fourchette de comparaison entre la sortie modèle et la sortie mesure de vitesse.

4 Limitations et précautions à prendre

4.1 Marche arrière du moteur

Comme expliqué dans le rapport, la marche arrière est source de bruit de vitesse. Le modèle de filtrage utilisé permet de filtrer au mieux ces bruits. Cependant, le modèle de toute la chaîne moteur dépend de l'état de charge de la batterie et l'ensemble des conditions de charge du rover (pente, charge utile...). La marche arrière est à utiliser donc avec précaution tant que le parallélisme des roues et la chaîne moteur ne sont pas modifiés. Une solution finale pour ce problème peut être :

1. Augmenter le rapport de réduction de vitesse de la chaîne de transmission.
2. Choisir un moteur qui permet de fournir le couple demandé en basse vitesse.

Une solution intermédiaire est la réduction de l'appel de couple au démarrage dans le sens négatif de rotation des roues. Pour ce faire, il suffit de régler le problème de parallélisme des roues.

En boucle fermée de commande le problème de changement du modèle peut être une source d'instabilité dangereuse pour le système. D'où le choix d'une boucle ouverte en commande par défaut.

4.2 Sortie centrale inertielle

La sortie centrale inertielle devient obsolète à cause des vibrations du moteur. Il est donc nécessaire de trouver une solution pour isoler la centrale ou estimer l'erreur introduit par la vibration (compliqué vu l'ordre de grandeur de l'erreur). Veuillez vous referez au rapport du stage pour plus d'informations.

4.3 Choix de batterie

La tension batterie maximale tolérable est de 8V. Cette limite est liée à l'alimentation du servo moteur qui est directement branché à la sortie batterie.