Concevoir un robot / Programmer les capteurs et actionneurs d'un robot

Cours correspondant : Robotique pratique (https://www.ensta-bretagne.fr/lebars/robotique_pratique.pdf).

Rendre sur Moodle dans un fichier de la forme DATE_NOM1_NOM2_..._NOMN.zip tous les documents utiles permettant de montrer le travail effectué : notes techniques, codes, fichiers de config, logs, données brutes et/ou traitées, infos précises sur les versions logicielles et matérielles utilisées, schémas en CAO ou manuels, captures d'écran, photos et vidéos à différentes étapes, etc.

Dans ce TD, **plusieurs activités à se repartir en équipes** de quelques personnes et à faire **à tour de rôle**, sont proposées :

1. Alimentation d'un robot : manipulations de batteries et chargeurs
2. Test d'un GPS
3. Test d'une AHRS
4. Test d'une carte d'interface capable de générer des signaux PWM pour contrôler des moteurs/servomoteurs
5. Utilisation de télécommandes, récepteurs, moteurs et servomoteurs
6. Test d'un autopilote
7. Test d'un autopilote simulé
8. Programmation en MATLAB, Python, Java, C/C++ des capteurs ou cartes d'interface

Tout le monde ne pourra a priori pas faire toutes les activités à cause de limites de matériel, temps, etc. l'idée est que selon ses connaissances, chacun puisse faire ce qu'il ne connaissait pas.

Activité 1 : Alimentation d'un robot : manipulations de batteries et chargeurs

Lieu : salle info
Nb max personnes : 4 binômes
Matériel nécessaire par équipe : chargeurs (peuvent être en plusieurs parties e.g. bloc d'alimentation séparé) et batteries divers (1 Li-Po 2S, 1 Li-Po 3S, 1 NiMH 5V,…) et leurs adaptateurs, checkers, 1 PC de salle info ou PC portable d'étudiant
Durée estimée : 30 min

Attention :

- **Ne pas confondre un bloc d'alimentation pour chargeur avec un chargeur !**
- **Ne pas confondre les entrées et les sorties du chargeur !**
- **Bien faire attention à ne pas créer un court-circuit dans la batterie notamment au moment du branchement de câbles adaptateurs sur les chargeurs, en particulier brancher d'abord les câbles au chargeur et en dernier la batterie pour minimiser les risques que les fiches bananes se touchent !**
- **Si les adaptateurs sont avec de multiples connecteurs, vérifier qu'ils ne risquent pas d'être court-circuités ou si autre chose y est déjà branché !**

Différents types de batteries sont à notre disposition. Le but de cette activité va être d'apprendre à charger différents types de batteries avec différents types de chargeurs en toute sécurité. Consulter https://youtu.be/vzAOG9ctZRU et si besoin la documentation du chargeur pour trouver comment le configurer, lancer une charge (ou « balance charge » si applicable), et l'interrompre au bout de 2 min, ceci pour chaque type de batterie (1 par 1) et avec au moins 2 modèles de chargeurs différents. **Vérifier avec votre encadrant avant de brancher la batterie**. Vérifier la tension des batteries avec un checker.

Activité 2 : Test d'un GPS

Lieu : salle info+dehors
Nb max personnes : 5 binômes
Matériel nécessaire par équipe : 1 GPS et adaptateurs USB éventuels et antenne externe éventuelle, 1 PC de salle info, 1 PC portable d'étudiant
Durée estimée : 45 min

Le but de cette activité est de savoir tester rapidement un GPS sur un PC. Les GPS fournis sont des Drotek Tiny RTK (nécessite une antenne externe) ou BU-353 qui se branchent en USB.

Sous Windows (avec un PC portable car il faudra aller dehors pour capter le GPS), une fois branché et l'installation du driver effectuée (driver à chercher sur Internet si besoin), il devrait apparaître sous un nom similaire à **COM1** dans la rubrique **Ports COM et LPT** du **Gestionnaire de périphérique** (on peut aussi modifier le numéro du port et d'autres options dans les propriétés avancées du port).

Télécharger, extraire, lancer **HyperTerminal**, à récupérer sur https://www.ensta-bretagne.fr/lebars/Share/hypertrm.zip . Répondre de la manière la plus logique aux éventuelles questions posées par le logiciel. Les paramètres importants à régler sont :

- **Baudrate** : **4800** ou **9600** en général pour un GPS

- **8 bits de données, pas de parité, 1 bit de stop, pas de contrôle de flux**

Cliquer sur les boutons **Connecter/Déconnecter** éventuellement plusieurs fois jusqu'à ce que des données s'affichent. Aller dans le menu **Transfert\Capturer le texte** pour enregistrer les données brutes. Un GPS classique utilise le protocole **NMEA** (voir e.g. https://archive.wikiwix.com/cache/index2.php?url=http%3A%2F%2Fwww.gpsinformation.org%2Fdale%2Fnmea.htm#federation=archive.wikiwix.com&tab=url). Observer les données et aller dehors (besoin d'une vue claire du ciel) pendant quelques minutes et observer l'évolution des données. Lorsque quelques satellites commenceront à être détectés, il devrait être possible de distinguer l'heure UTC dans les données, puis la latitude et la longitude. Eventuellement utiliser **Google Earth Pro** pour avoir une idée de la latitude et la longitude à Brest. Les latitudes et longitudes peuvent être exprimées dans des unités légèrement différentes, utiliser e.g. https://www.ensta-bretagne.fr/lebars/utilities/GPSDataConverter_vs2008sp1.zip pour faire des conversions rapides. Utiliser e.g. les outils dans https://github.com/ENSTABretagneRobotics/File_conversions (notamment NMEA2CSV et lognav2KML) pour convertir les données brutes et pouvoir afficher le trajet que vous avez fait dehors dans Google Earth Pro (trames NMEA du GPS à convertir en fichier KML pour Google Earth Pro).

Utiliser un PC de salle info ou perso sous Linux pour tester rapidement de manière similaire le périphérique sous Linux (juste vérifier rapidement qu'on peut envoyer/recevoir quelques données). Le nom du périphérique sera similaire à **/dev/ttyUSB0** ou **/dev/ttyACM0** ou **/dev/ttyAMA0** ou **/dev/ttyS0** et un équivalent possible de **HyperTerminal** peut être **minicom** ou **gtkterm**.

Activité 3 : Test d'une AHRS

Lieu : salle info
Nb max personnes : 8 binômes
Matériel nécessaire par équipe : 1 Razor et adaptateurs ou câbles USB, 1 PC de salle info, 1 PC portable d'étudiant
Durée estimée : 60 min

Le but de cette activité est de tester et calibrer une AHRS sur un PC. Les AHRS fournies sont des SparkFun Razor (modèles **SEN-10736** (voir https://web.archive.org/web/20241203081100/https://www.sparkfun.com/products/retired/10736) pour la version sans USB, **M0**, **OpenLog Artemis**) qui se branchent en USB directement ou via un câble convertisseur **FTDI TTL-232R-3V3** (**ce câble fournit une alimentation 5 V**, voir https://www.ftdichip.com/Support/Documents/DataSheets/Cables/DS_TTL-232R_CABLES.pdf, des pins à souder sont fournies si besoin, aller en M003 pour souder si besoin).

Attention : vérifier les branchements avec vos encadrants avant de connecter le câble convertisseur au PC la première fois !

Sous Windows, une fois branchée et l'installation du driver effectuée, elle devrait apparaître sous un nom similaire à **COM1** dans la rubrique **Ports COM et LPT** du **Gestionnaire de périphérique** (on peut aussi modifier le numéro du port et d'autres options dans les propriétés avancées du port).

Télécharger, extraire, lancer **HyperTerminal**, à récupérer sur https://www.ensta-bretagne.fr/lebars/Share/hypertrm.zip . Répondre de la manière la plus logique aux éventuelles questions posées par le logiciel. Les paramètres importants à régler sont :

- **Baudrate** : **à trouver**, dépend du firmware installé (a priori entre 4800 et 115200) !
- **8 bits de données, pas de parité, 1 bit de stop, pas de contrôle de flux**

Cliquer sur les boutons **Connecter/Déconnecter** éventuellement plusieurs fois jusqu'à ce que des données s'affichent. Aller dans le menu **Transfert\Capturer le texte** pour enregistrer les données brutes. Les données qui s'affichent dépendent de la version du firmware, il se peut aussi que les données soient envoyées sur un autre port que celui qu'on utilise, ou encore que le firmware ne renvoie rien par défaut...

Quel que soit le firmware déjà installé, nous allons le changer. Suivre globalement les instructions sur https://github.com/ptrbrtz/razor-9dof-ahrs/wiki/Tutorial pour changer le firmware (la dernière version du firmware sera à prendre sur https://github.com/lebarsfa/razor-9dof-ahrs) et faire les calibrations proposées. A priori, on pourra utiliser **Arduino IDE** sur les PC des salles info sous Windows (si besoin de télécharger certains fichiers pour tester sur PC perso, voir depuis un PC de salle info (disques réseau) les dossiers **Arduino** et **Processing** disponibles sur **enseignement\rob\...\Robotique pratique** (regarder dans **public\share** si **enseignement\rob** n'est pas disponible)). On peut vérifier si la calibration des accéléromètres est bonne en donnant du roulis/tangage à cap constant : le cap retourné ne doit pas trop varier quand le roulis/tangage devient important. On peut aussi vérifier la calibration magnétique en tournant l'AHRS de 90 degrés 4 fois et en vérifiant qu'elle a bien indiqué à chaque fois une différence de 90 degrés.

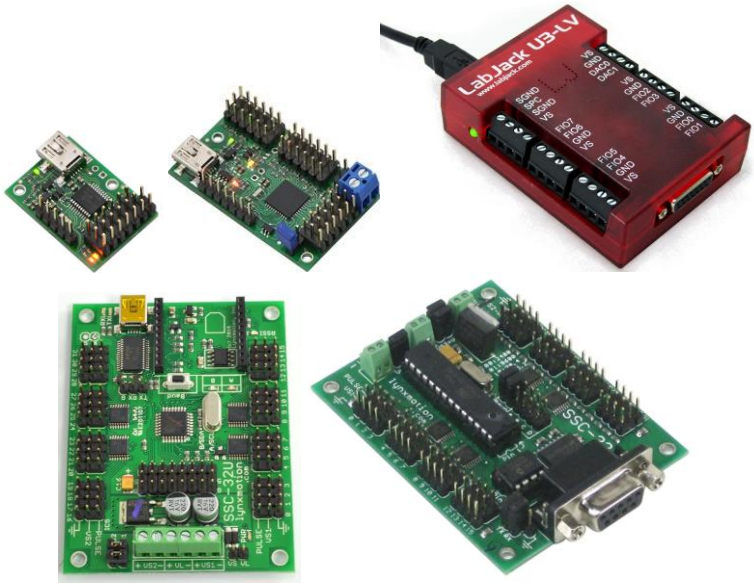
Utiliser un PC de salle info ou perso sous Linux pour tester rapidement de manière similaire le périphérique sous Linux (juste vérifier rapidement qu'on peut envoyer/recevoir quelques données, si nécessaire télécharger **Arduino IDE**). Le nom du périphérique sera similaire à **/dev/ttyUSB0** ou **/dev/ttyACM0** ou **/dev/ttyAMA0** ou **/dev/ttyS0** et un équivalent possible de **HyperTerminal** peut être **minicom** ou **gtkterm**.

Activité 4 : Test d'une carte d'interface capable de générer des signaux PWM pour contrôler des moteurs/servomoteurs

Lieu : salle info

Nb max personnes : 6 binômes
Matériel nécessaire par équipe : les différentes cartes et adaptateurs ou câbles USB, 2 servomoteurs, 1 batterie 5 V (ou BEC/régulateur 5 V au moins 1 A avec une autre batterie), 1 PC de salle info, 1 PC portable d'étudiant
Durée estimée : 30 min

Le but de cette activité est de savoir tester une carte d'interface pour PC pour pouvoir contrôler des servomoteurs. Les cartes fournies sont des **Lynxmotion SSC-32/SSC-32u** ou **Polulu Maestro** ou **LabJack U3** et se branchent en **USB** via des câbles ou convertisseurs à déterminer.



Bien consulter la documentation sur Internet pour comprendre comment alimenter la carte et les servomoteurs pour les tests.

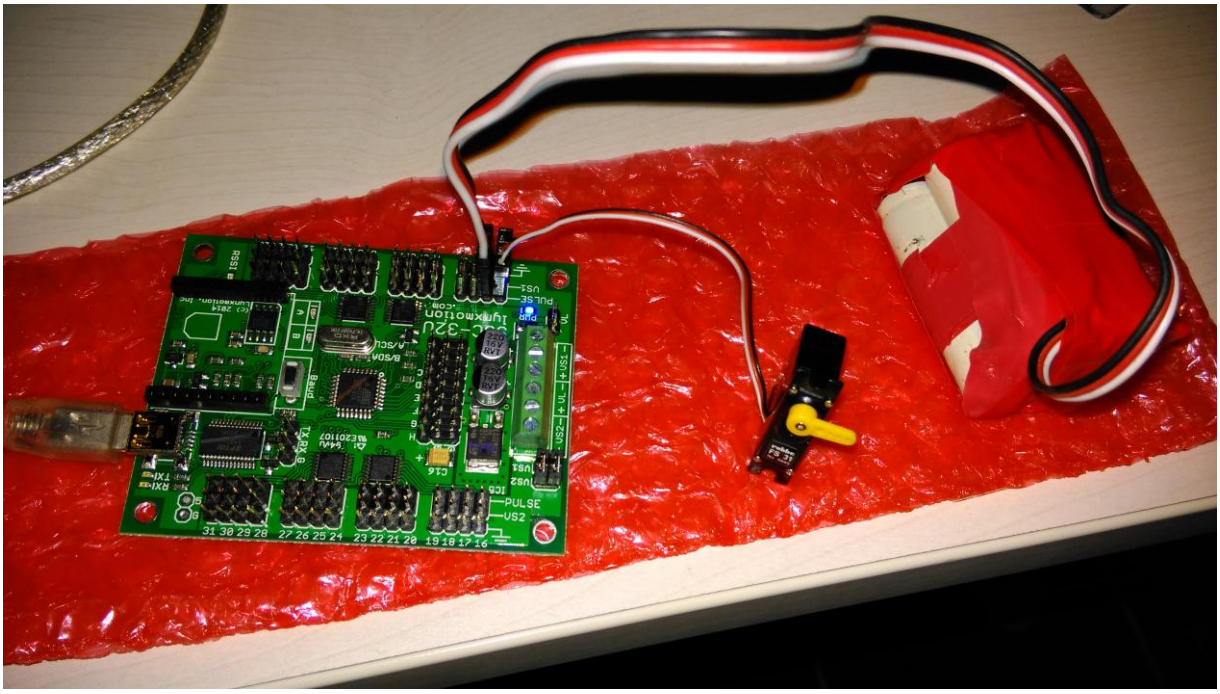
Sous Windows, une fois branchée et l'installation du driver effectuée, elle devrait apparaître sous un nom similaire à **COM1** dans la rubrique **Ports COM et LPT** du **Gestionnaire de périphérique** (on peut aussi modifier le numéro du port et d'autres options dans les propriétés avancées du port).

Sous Linux (PC hors salle info requis pour Pololu) le nom du périphérique sera similaire à **/dev/ttyUSB0** ou **/dev/ttyACM0** ou **/dev/ttyAMA0** ou **/dev/ttyS0**.

Pour les cartes **Lynxmotion**, télécharger, extraire, lancer https://www.ensta-bretagne.fr/lebars/Share/hypertrm.zip pour Windows ou **minicom** ou **gtkterm** pour Linux. Répondre de la manière la plus logique aux éventuelles questions posées par le logiciel. Les paramètres importants à régler sont :

- **Baudrate** : **à trouver** (a priori entre 4800 et 115200) !
- **8 bits de données, pas de parité, 1 bit de stop, pas de contrôle de flux**

Brancher 1 ou 2 servos et les faire bouger. Consulter la documentation pour trouver quelles commandes doivent être tapées.



Pour les cartes Pololu, le logiciel du constructeur sera nécessaire pour tester les servos, le lancer ou l'installer si nécessaire.

Pour les cartes LabJack, le nécessaire est normalement installé sur les PC des salles infos sous Windows, le logiciel du constructeur sera nécessaire pour tester les servos. On pourra notamment utiliser l'application **LJControlPanel** et sa fonction **Test** pour configurer un ou plusieurs **Timer**/**PWM16**. Consulter la documentation pour plus d'informations… Aide :

Le LabJack peut générer jusqu'à 6 Timers/PWM dont la fréquence est définie par

Fréquence finale du PWM
Doit être proche de 1/20ms
pour pouvoir contrôler un servo

Fréquence interne du LabJack

$$f_{PWM} = \frac{f_{syst}}{timer\_clock\_divisor * 2^{16}}$$

Paramètre pouvant être propre à chaque PWM
pour faire varier sa fréquence indépendamment des autres

Mode du PWM
=> précision de la largeur d'impulsion

Attention à bien les configurer pour que la fréquence du PWM et la largeur d'impulsion soient valides pour des servomoteurs standards.

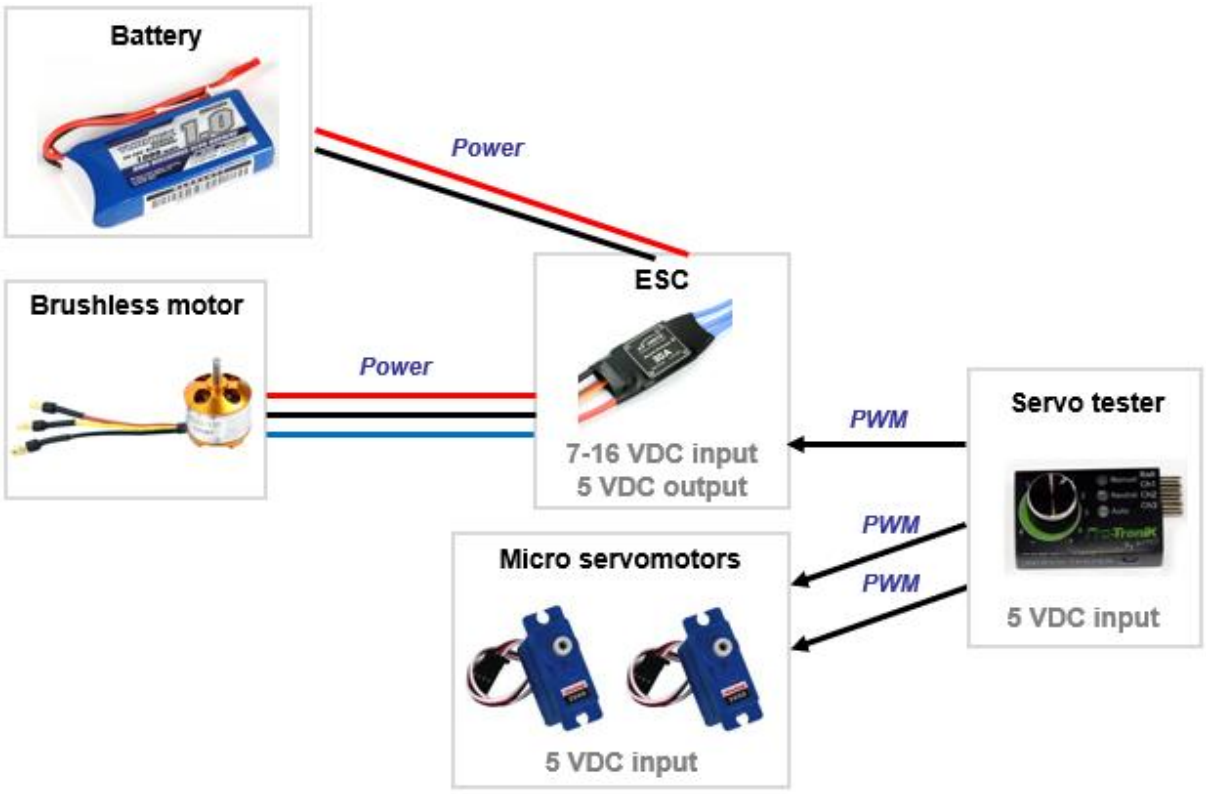
Activité 5 : Utilisation de télécommandes, récepteurs, moteurs et servomoteurs

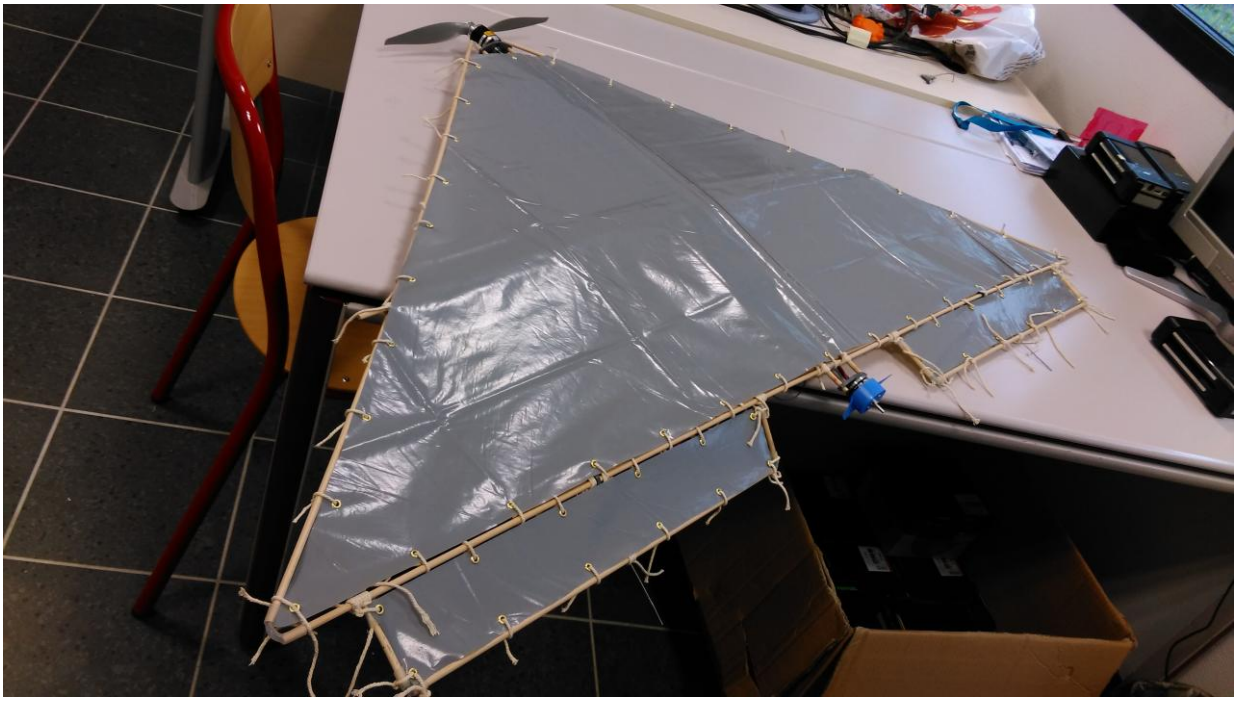
Lieu : salle info

Nb max personnes : 4 trinômes
Matériel nécessaire par équipe : 1 moteurs brushless, 1 ESC, 3 connecteurs PK3.5 (si l'ESC ne les a pas déjà), 1 connecteur JST mâle et 1 femelle (si l'ESC ne les a pas déjà), 1 batterie avec connecteur JST femelle et de tension compatible avec l'ESC et le moteur, 1 batterie 5 V ou BEC/régulateur 5 V au moins 1 A (si l'ESC n'a pas de BEC 5 V), 2 servomoteurs, 1 testeur de servos, 1 télécommande avec sa batterie et son récepteur, 1 PC de salle info
Durée estimée : 90 min

Le but de cette activité est de savoir tester rapidement des moteurs et manipuler des ESC, BEC, testeurs de servos, récepteurs et télécommandes.

Faire fonctionner le montage suivant (voir infos qui suivent pour les soudures éventuelles à faire), correspondant aux actionneurs d'une aile delta :





Le testeur de servo permettra de faire tourner les moteurs et servomoteurs à une vitesse/position donnée. Il faudra bien veiller à choisir des éléments compatibles parmi ceux à votre disposition. Certains connecteurs de servos peuvent avoir une forme incompatible avec le testeur, rogner la partie du connecteur gênante si nécessaire pour respecter le format JR.

Des connecteurs dorés PK 3.5 devront peut-être être soudés sur l'ESC, pour être compatibles avec ceux déjà présents sur le moteur (aller en M003 pour souder si besoin).

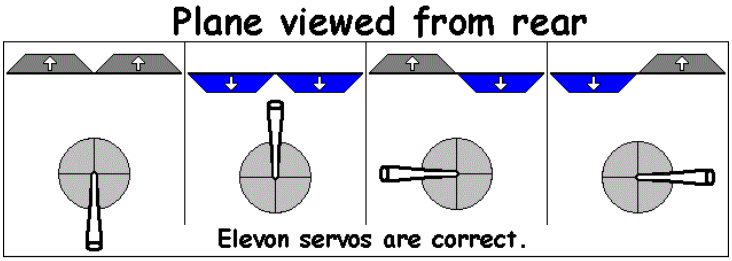
De plus, 1 connecteur JST femelle étant présent sur la batterie en photo, il faudra peut-être souder un connecteur JST mâle sur l'entrée batterie de l'ESC. De plus, pour permettre l'alimentation d'autres éléments optionnels, 1 connecteur JST femelle devra être soudé en parallèle du mâle.

Ne pas oublier de passer des gaines thermorétractables (pour isoler les soudures et éviter les court-circuits) avant de souder et faire attention à ce qu'elles ne se rétractent pas pendant la soudure à cause de la chaleur qui se propage dans les câbles !

Attention : aller progressivement et vérifier avec vos encadrants avant de brancher la batterie la première fois ou si vous avez un doute !

Remplacer ensuite le testeur de servo par un récepteur de modélisme et configurer la télécommande (trouver sa documentation sur Internet) pour contrôler de manière pratique une aile delta : moteur+1 servo pour aileron arrière droit+1 servo pour aileron arrière gauche. Ceci implique de configurer un mixage de voies (créer un nouveau « MODEL » dans la télécommande pour ne pas modifier les existants s'il y en a) :

- Manche vers le bas : les 2 ailerons vers le haut
- Manche vers le haut : les 2 ailerons vers le bas
- Manche vers la droite : 1 aileron en haut, l'autre en bas
- Manche vers la gauche : inversement



Le moteur sera contrôlé sur le 2ème manche (celui qui ne revient pas automatiquement au milieu) :

- Manche vers le bas : moteur à l'arrêt
- Manche vers le haut : pleine puissance

Note : il faudra a priori consulter la documentation du récepteur et suivre la procédure de bind pour associer le récepteur à l'émetteur de la télécommande (attention : tous les récepteurs ne sont pas forcément identiques, de même certaines télécommandes ont des récepteurs intégrés, d'autres sont amovibles, etc.). De plus, il faudra aussi configurer la télécommande elle-même pour réaliser le mixage de voies, et peut-être pour certaines options particulières liées à son émetteur.

Activité 6 : Test d'un autopilote

Lieu : salle info
Nb max personnes : 4 trinômes

Matériel nécessaire par équipe : 1 autopilote (APM2.5, HKpilot Mega, Pixhawk, HKpilot32, Dropix v2, RadioLink Mini Pix, Drotek Pixhawk 3 Pro ou Pixhawk 4 Mini, etc.) et ses accessoires et son câble USB, 1 GPS compatible (idéalement avec boussole I2C externe intégrée), 1 kit de télémétrie 433 MHz ou 2.4 GHz et son câble micro USB, 2 servomoteurs (ou 1 servomoteur et 1 moteur+ESC+batterie compatible), 1 batterie 5 V (ou BEC/régulateur 5 V au moins 1 A avec une autre batterie), 1 testeur de servos et/ou 1 télécommande avec sa batterie et son récepteur, 1 PC de salle info sous Windows, 1 PC portable d'étudiant sous Windows, 1 robot roulant de type buggy, voiture ou char (à aller chercher à la fin)
Durée estimée : 90 min ou plus

Attention à bien vérifier dans les documentations des divers éléments si les connecteurs doivent être modifiés car il est courant que d'un modèle à l'autre les connecteurs ne soient pas branchés de la même manière même si les connecteurs sont les mêmes…

Le but de cette activité est de savoir utiliser un autopilote ArduPilot ou compatible : calibration, test avec des servomoteurs avec télécommande+récepteur ou testeur de servos.

Un autopilote est un regroupement de plusieurs capteurs et cartes d'interfaces utiles pour un robot : magnétomètres, gyromètres, accéléromètres, lecture et génération de signaux PPM, SBUS ou PWM (pour s'interfacer avec un récepteur de modélisme, des servomoteurs, des ESCs), peut être facilement interfacé à un GPS ou d'autres capteurs et modules de communication. Un autopilote compatible avec la famille de firmware ArduPilot (ArduCopter pour les multirotors, ArduPlane pour les avions, ArduRover pour les robots roulants et bateaux à moteur, ArduSub pour le sous-marin BlueROV, etc.) fournit en plus un certain nombre d'algos de contrôle potentiellement utiles.

En résumé, l'installation d'un autopilote compatible ArduPilot implique en général les étapes suivantes :

1. Branchement en USB à un PC pour installer le firmware souhaité à l'aide de Mission Planner.
2. Branchement des périphériques souhaités sur l'autopilote (e.g. récepteur, GPS, télémétrie, servos, ESCs). Garder l'autopilote hors tension pendant les branchements.
3. Calibration des capteurs, configuration des paramètres pour indiquer notamment quel canal de la télécommande doit correspondre à quelle fonction ou quel actionneur à

l'aide de Mission Planner (des paramètres de configuration de la télécommande devront aussi peut-être être changés selon ce qui est souhaité, attention aux éventuelles différences de numérotation des canaux entre l'autopilote, le récepteur et la télécommande). En particulier, il faudra réfléchir à quels modes de fonctionnement de l'autopilote peuvent être utiles (e.g. **Manual**, **Auto**, **RTL**).

4. Chargement éventuel de waypoints que l'autopilote devra suivre quand on le passe en mode **Auto**.
5. Préarmement de l'autopilote (bouton rouge clignotant si disponible) ou paramètre spécifique si le préarmement doit être toujours autorisé.
6. Armement de l'autopilote. Diverses vérifications vont être faites (e.g. calibrations bien effectuées, qualité de réception du GPS), certaines pouvant être désactivées via paramètres.
7. Utilisation. Selon le mode choisi, la télécommande contrôle plus ou moins directement les actionneurs.
8. Désarmement.

6.1 Vérification de la communication sur un PC

Sous Windows, une fois branché en USB et l'installation du driver effectuée (installer https://firmware.ardupilot.org/Tools/MissionPlanner/driver.msi si Windows Update ne trouve pas le driver automatiquement), l'autopilote devrait apparaître sous un nom similaire à **COM1** dans la rubrique **Ports COM et LPT** du **Gestionnaire de périphérique** (on peut aussi modifier le numéro du port et d'autres options dans les propriétés avancées du port).

Note : bien souvent l'autopilote apparait sous un nom temporaire dans les premières secondes. C'est en fait son bootloader qui se lance au début et qui attend pendant quelques secondes une commande spéciale éventuelle qui lui demanderait de se placer en attente d'un nouveau firmware. Sinon, il lance son firmware actuel, qui sera alors détecté potentiellement sous un (ou même plusieurs, selon les fonctionnalités qu'il propose) autre nom dans le **Gestionnaire de périphérique**.

Télécharger, extraire, lancer **HyperTerminal**, à récupérer sur https://www.ensta-bretagne.fr/lebars/Share/hypertrm.zip . Répondre de la manière la plus logique aux éventuelles questions posées par le logiciel. Les paramètres importants à régler sont :

- **Baudrate** : **115200**
- **8 bits de données, pas de parité, 1 bit de stop, pas de contrôle de flux**

Cliquer sur les boutons **Connecter**/**Déconnecter** éventuellement plusieurs fois jusqu'à ce que des données s'affichent. Aller dans le menu **Transfert\Capturer le texte** pour enregistrer les données brutes. Les données qui s'affichent ne sont pas toutes humainement compréhensibles, le protocole utilisé est le **MAVLink**, l'idée est juste de vérifier rapidement que l'autopilote peut bien communiquer avec le PC (on suppose ici que le firmware actuellement sur l'autopilote utilise le protocole MAVLink, ce qui est normalement le cas par défaut pour les modèles à disposition).

6.2 Configuration avec Mission Planner et premiers tests sur table du contrôle des actionneurs par télécommande via l'autopilote avec ArduRover en mode Manual

Suivre approximativement la liste des étapes précédemment citées pour arriver à contrôler un servomoteur et un moteur (utiles e.g. pour un modèle type voiture, si moteur+ESC indisponible utiliser éventuellement à la place un autre servomoteur) par télécommande via l'autopilote avec ArduRover en mode Manual. Les infos des paragraphes suivants devraient aider à y arriver.

Attention à surveiller l'état des batteries pour ne pas trop les décharger !

Bien consulter la documentation sur Internet pour comprendre comment alimenter le type d'autopilote utilisé et les servomoteurs pour les tests (voir e.g. http://ardupilot.org/rover/docs/common-powering-the-apm2.html , http://ardupilot.org/rover/docs/common-pixhawk-wiring-and-quick-start.html , https://drotek.gitbook.io/dropix-user-guide/ , https://drotek.gitbook.io/pixhawk-3-pro/ , https://shop.holybro.com/pixhawk4-mini_p1120.html , etc.). Enlever le jumper JP1 des APM/HKpilot Mega pour commencer…

Receiver power consumption: there is usually a RC row on the autopilot that is supposed to be connected to the receiver (most of the autopilots need to receive PPM or SBUS signals from the receiver, note also that PPM might be used also as communication between an external transmitter and the JR module of a radio). Autopilots such as the Pixhawk/HKPilot32 are supposed to provide 5 V to the receiver through this row, but with a very limited current. For powerful receivers, it is better to avoid using the 5 V coming from the RC row from the autopilot and use a 5 V coming from elsewhere (e.g. a BEC or a dedicated 5 V battery) instead.

Télécharger, extraire, lancer **Mission Planner** (if it is not already installed, get it from http://firmware.ardupilot.org/Tools/MissionPlanner/MissionPlanner-latest.msi (installer only for Windows), https://www.ensta-bretagne.fr/lebars/Share/MissionPlanner-latest.zip (should work for Windows with https://dotnet.microsoft.com/download/dotnet-framework/net48 , Ubuntu with https://www.mono-project.com/download/stable/), https://github.com/ArduPilot/MissionPlanner/releases/tag/osxlatest (only for macOS), note that some functionalities of Mission Planner might not be available on Linux or macOS), alternatively **QGroundControl** is similar to **Mission Planner** on Linux or macOS).

Suivre l'assistant de configuration pour charger le firmware **ArduRover** (voir http://ardupilot.org/rover/index.html pour plus d'infos, notamment les points principaux dans les sections en rouge, voir aussi la note de vocabulaire à la fin de l'activité) et faire les calibrations demandées. L'autopilote ayant peut-être été déjà configuré pour d'autres choses, il est recommandé de restaurer les paramètres par défaut juste après le chargement du firmware, puis redémarrer l'autopilote : après connexion à l'autopilote dans Mission Planner (choisir le bon port et bon baudrate (en général 115200 quand connecté via USB, 57600 via télémétrie) puis cliquer sur **CONNECT** en haut à droite), cliquer sur **Reset to Default** dans **CONFIG\Full Parameter List**, attendre 30 s, puis débrancher-rebrancher l'autopilote.

Note : certaines fonctionnalités du firmware peuvent être bridées selon la version matérielle de l'autopilote (e.g. RAM <= 256 KB). Problèmes de compatibilité connus :

- APM2.5 et similaires : le firmware autour de v3.2.1 doit être le dernier supporté.
- RadioLink Mini Pix, Drotek Dropix, Pixhawk v1 et similaires ont probablement des limitations à partir des firmware autour de v4.0.0. Parfois il est préférable d'utiliser

une version inférieure du firmware car certaines fonctionnalités existantes peuvent être désactivées en plus de certaines nouvelles.
- Drotek Pixhawk 3 Pro : voir https://github.com/ArduPilot/ardupilot/issues/21184.
- Si besoin, aller dans **SETUP\Install Firmware\All Options** pour forcer une version spécifique. Il faudra alors bien faire attention à choisir le bon modèle dans **Platform** (attention : il y a un risque que l'autopilote devienne inutilisable (« bricked ») en cas d'erreur si le bootloader de l'autopilote est modifié).

Le récepteur doit être configuré pour sortir des signaux PPM ou SBUS (et dans les firmware ArduPilot, le signal PPM ou SBUS venant du récepteur se branche sur la même pin de l'autopilote (souvent appelée PPM IN ou RCIN) et le protocole est déterminé automatiquement) au lieu de PWM, consulter sa documentation. Si télécommande+récepteur indisponibles, tester avec un joystick (voir activité avec un autopilote simulé, attention aux versions d'ArduRover < V4.0.0 qui ne supportent peut-être pas le joystick), ou au pire désactiver le **RC channel arming check** dans **CONFIG\Standard params**.

Il se peut que l'autopilote se mette en sécurité si tous les canaux de la télécommande et les actionneurs nécessaires selon le mode ne sont pas branchés, de plus il faut peut-être (étape de préarmement) appuyer sur le bouton rouge clignotant (à brancher sur l'autopilote si dispo) jusqu'à ce qu'il devienne fixe, ou désactiver le paramètre **BRD_SAFETY_DEFLT** (**BRD_SAFETYENABLE** dans ArduRover < V4.4.0) et redémarrer l'autopilote avant de pouvoir armer. L'onglet **DATA\Messages** dans Mission Planner peut aider à déterminer ce qui bloque l'armement. L'onglet **DATA\Actions** contient un certain nombre de boutons utiles.

6.3 Tests en extérieur

Si disponible, configurer (bind/appairage éventuels, voir https://ardupilot.org/copter/docs/common-configuring-a-telemetry-radio-using-mission-planner.html…) aussi une paire de modules de télémétrie pour mettre en place une communication sans fil (bien vérifier les branchements sur la documentation de l'autopilote et celle du module de télémétrie, il se peut que les connecteurs doivent être modifiés). Le but final sera que Mission Planner puisse communiquer sans fil avec l'autopilote. Si besoin, HyperTerminal peut être dans un premier temps utilisé pour tester s'il est possible d'envoyer et recevoir des caractères via ce type de liaison entre 2 ordinateurs : brancher un module de télémétrie en USB sur chaque ordinateur, lancer HyperTerminal sur chaque ordinateur et ouvrir le bon port au baudrate supporté par le module de télémétrie (57600 en général) et vérifier que quand on tape un caractère sur l'un, il apparait sur l'autre.

Si de plus un GPS compatible est disponible (bien vérifier les branchements sur la documentation de l'autopilote et celle du GPS, il se peut que les connecteurs doivent être modifiés), aller dehors (pour capter le GPS) et vérifier que Mission Planner indique bien votre position. A noter que si on n'a pas de GPS compatible mais qu'on a un autre type de capteur capable de donner des positions, on peut les envoyer à l'autopilote en s'inspirant des infos dans https://www.ensta-bretagne.fr/lebars/Share/GPS_INPUT_pymavlink.py.

6.4 Tests de suivi de points GPS sur un robot

Une fois les premiers tests réussis, récupérer un buggy, voiture ou char sans électronique (mais quand même muni de moteurs+ESCs+batterie et servomoteurs éventuels) et y installer

l'autopilote et tout le nécessaire, le tester d'abord en téléguidé (pour les robots se contrôlant comme des chars, on peut utiliser un V-tail mixer ou modifier les paramètres **SERVO1_FUNCTION** = 73 (Throttle Left), **SERVO3_FUNCTION** = 74 (Throttle Right) (voir aussi **PILOT_STEER_TYPE** et **PIVOT_TURN_ANGLE** ?) pour convertir les signaux PWM moteur et direction en signaux PWM moteur droit et moteur gauche) et vérifier que le cap et la position GPS sont corrects dans Mission Planner puis tester la fonction **Fly to here** (clic droit sur la carte dans Mission Planner, ceci placera l'autopilote en mode **Guided**). Les paramètres de PID dans **CONFIG\Basic Tuning** devront peut-être être ajustés selon la dynamique du robot.



6.5 Vocabulaire

Note de vocabulaire (ou plutôt noms des différents logiciels) dans le cas d'un drone de type avion :

- Mission Planner est un logiciel GCS (Ground Control Station) fait pour Windows (fonctionne aussi en grande partie sur Ubuntu) permettant de configurer un autopilote comme le Pixhawk (changement du firmware, configuration de paramètres et notamment des waypoints, affichage de l'état du Pixhawk, changements de modes, envoi de scripts particuliers, etc.). Il est relativement indispensable pour préparer le Pixhawk avant un vol. Bien que pratique pour suivre l'état du vol et faire des changements divers, il n'est pas strictement indispensable pendant un vol si tout a été bien préparé à l'avance, la télécommande (et surtout le Pixhawk via les paramètres correspondants) pouvant être configurée pour avoir des interrupteurs permettant des changements divers (notamment passage de mode manuel à auto, armement, désarmement, Return To Launch, etc.).
- ArduPlane est le nom du firmware typiquement utilisé pour un avion (dans la famille des firmware ArduPilot, il y a aussi notamment ArduCopter pour les hélicoptères et multirotors, ArduRover pour les robots roulants et bateaux à moteur) et ce firmware s'exécute sur le Pixhawk. L'essentiel de l'intelligence est dedans (c'est un OS temps réel probablement proche d'un Linux très allégé, avec des bibliothèques et algos pour gérer les capteurs, actionneurs, etc.). On peut communiquer avec lui avec le protocole MAVLink, et c'est ce que fait Mission Planner. Un programme sur e.g. une Raspberry Pi ou tout autre chose peut aussi communiquer avec le Pixhawk ou Mission Planner ou les 2 et/ou tout autre programme utilisant le protocole MAVLink. A noter cependant que les uns et les autres ne supportent qu'un sous ensemble du protocole MAVLink, https://www.ensta-bretagne.fr/lebars/Share/fake_autopilot.zip ne supporte par exemple que très peu de choses, alors qu'ArduPlane supporte a priori les choses documentées sur https://ardupilot.org/dev/docs/mavlink-commands.html . Il y a quelques infos/exemples (principalement pour ArduRover) dans les activités qui suivent. Une manière de contrôler un peu le Pixhawk indépendamment d'une communication MAVLink est via les signaux PWM (ou en fait leur version sérialisée sur un seul fil,

en PPM ou SBUS sur le port RCIN du Pixhawk) qu'on lui envoie via récepteur+télécommande, l'interprétation de ces signaux dépendant de divers paramètres réglables dans le Pixhawk (e.g. https://ardupilot.org/plane/docs/parameters.html#rc10-option-rc-input-option).

Activité 7 : Test d'un autopilote simulé

Lieu : salle info
Nb max personnes : illimité
Matériel nécessaire par équipe : 1 PC de salle info ou PC portable d'étudiant
Durée estimée : 60 min

Pour la prise en main des fonctions avancées d'autonomie des autopilotes ArduPilot ou compatible, une version spéciale des firmware ArduRover et ArduCopter existe en mode simulé (voir l'activité sur l'autopilote réel pour plus d'infos sur les autopilotes, ArduPilot, Mission Planner). Parmi les différences avec un réel, il y a :

- On ne peut a priori pas faire les calibrations des centrales inertielles, donc il faut potentiellement **désactiver** certains **Arm Checks** (dans **CONFIG\Standard Params**). Penser aussi à sélectionner e.g. **Quad** dans **SETUP\Frame Type** pour ArduCopter (comme pour un réel).
- On ne peut pas brancher de récepteur, donc on utilise un joystick (qui peut être réel, ou virtuel avec **vJoy Feeder**, voir **DATA\Actions\Joystick** dans Mission Planner pour activer et configurer un joystick (semble non disponible sous Linux)). On peut armer/désarmer avec le bouton **Arm/Disarm** dans l'onglet **DATA\Actions** (set the layout of Mission Planner to Advanced if it does not appear). Une solution alternative peut être d'envoyer des commandes via **MAVProxy** (e.g. **mavproxy.py --master=tcp:127.0.0.1:5760** and send **arm throttle** and **rc 3 1500** commands).
- Au lieu d'être reconnu comme un port série via USB (e.g. COM1), l'autopilote simulé va créer un **serveur TCP** sur le port **5760** (ainsi que **5762** et **5763** si un programme s'est déjà connecté sur le 5760).
- On peut changer le paramètre **SIM_GPS_DISABLE** pour simuler un vol en intérieur où on ne capte pas le GPS, d'autres paramètres similaires spécifiques à la simulation existent.

A l'aide des documents sur https://www.ensta-bretagne.fr/lebars/Share/ArduPilot%20simulator.zip (ou via le menu **SIMULATION** de Mission Planner) et en s'inspirant de ce qu'il y a à faire dans l'activité avec l'autopilote réel, tester un quadrirotor simulé avec le firmware **ArduCopter** (voir http://ardupilot.org/copter/index.html pour plus d'infos (notamment les points principaux dans les sections en rouge)) dans les modes **Stabilize**, **AltHold**, **Loiter**, **Land**, **RTL** et effectuer une mission de suivi de waypoints automatique.

Pour info, voici un exemple de check-list qu'on suivait parfois pour la configuration d'un drone aérien : https://www.ensta-bretagne.fr/lebars/Share/drone_take-off_check-list.txt

Activité 8 : Programmation en MATLAB, Python, Java, C/C++ de capteurs ou cartes d'interface

Lieu : salle info
Nb max personnes : illimité
Matériel nécessaire par équipe : 1 PC de salle info ou PC portable d'étudiant
Durée estimée : variable

Le but de cette activité est de savoir récupérer les données de capteurs et/ou contrôler des cartes d'interface dans un langage de programmation (MATLAB, Python, Java, C++). Le matériel proposé est :

- Centrale inertielle Razor pour récupérer les angles de roulis, tangage, lacet.
- GPS USB pour récupérer la latitude, longitude, altitude.
- Cartes d'interface Lynxmotion SSC-32/SSC-32u ou Polulu Maestro capables de générer des signaux PWM pour contrôler des moteurs/servomoteurs.
- Autopilotes ArduPilot ou compatibles pour récupérer les angles de roulis, tangage, lacet, la latitude, longitude, altitude si un GPS est disponible, les signaux PWM d'un récepteur si disponible, ainsi que contrôler des servomoteurs (il est en général plus simple de tester d'abord avec un firmware ArduRover car il y a moins de réglages de sécurité à désactiver par rapport à ArduCopter, cependant certaines fonctionnalités peuvent ne pas être disponibles).
- Le capteur de flux optique PX4FLOW communique ses données d'une manière similaire aux autopilotes, sauf que ce sont les vitesses instantanées vx,vy qui peuvent être intéressantes.

A noter que la **partie Java n'est plus maintenue**.

1. Si vous ne l'avez pas déjà fait, consulter les activités précédentes pour être déjà capable de voir les données brutes avec un logiciel tout fait.

2. Trouver comment utiliser https://github.com/ENSTABretagneRobotics/Hardware-MATLAB, https://github.com/ENSTABretagneRobotics/Hardware-Python, https://github.com/ENSTABretagneRobotics/Hardware-Java pour pouvoir contrôler votre matériel en MATLAB, Python ou Java (seulement MATLAB pour autopilotes et PX4FLOW).

3. Le code fourni faisant appel en interne à une bibliothèque écrite en C (dispo sur https://github.com/ENSTABretagneRobotics/Hardware-CPP), il y a un certain nombre de limitations. Pour avoir plus de contrôle sur votre matériel, il peut être plus intéressant de n'utiliser que des fonctions MATLAB, Python, Java, C/C++ « classiques ».

MATLAB

A l'aide de http://fr.mathworks.com/help/matlab/serial-port-devices.html, écrire un programme purement MATLAB capable de faire à peu près la même chose que le code fourni précédemment : récupération de yaw, pitch, roll pour la Razor, récupération de la latitude et longitude pour un GPS, commande de servomoteurs pour la Lynxmotion SSC-32/SSC-32u ou Polulu Maestro. Un code à trous avec des indications est fourni (**RazorAHRSMATLABTest_students.m** pour le cas de la Razor, s'en inspirer aussi pour les autres cas). Les fonctions **serial**, **set**, **fopen**, **fprintf**, **fscanf**, **fread**, **fwrite**, **fclose**, **sprintf**, **sscanf**, **strfind**, **disp**, **delete**, **clear** pourront notamment être utiles.

Si à un moment le port série n'est pas correctement fermé, utiliser la commande **fclose(instrfind)** pour pouvoir de nouveau y avoir accès.

Python

Il faut probablement utiliser les fonctions de **pyserial**.

Pour le cas de l'ArduPilot le protocole **MAVLink** est utilisé pour la communication entre l'autopilote et Mission Planner ou tout autre application. En Python, on peut utiliser **pymavlink**, mais il est assez peu documenté, voir les exemples https://www.ensta-bretagne.fr/lebars/Share/rcin_rover_pymavlink.py (voir https://www.ensta-bretagne.fr/lebars/Share/rcin_mavlink.pdf pour le tester en simulation), https://www.ensta-bretagne.fr/lebars/Share/rcoverride_rover_pymavlink.py, https://www.ensta-bretagne.fr/lebars/Share/manualcontrol_rover_pymavlink.py, https://www.ensta-bretagne.fr/lebars/Share/guided_rover_pymavlink.py, https://www.ensta-bretagne.fr/lebars/Share/nogps_takeoff_land_pymavlink.py, ainsi que l'activité suivante, la logique, les types, constantes et noms de fonctions du protocole MAVLink sont similaires en C/C++ et en Python (certaines fonctions sont aussi en plus dans pymavlink).

Java

La bibliothèque Java **PureJavaComm** (voir http://www.sparetimelabs.com/purejavacomm/purejavacomm.php) qui utilise en interne la bibliothèque **JNA** (voir https://github.com/java-native-access/jna) permet de programmer en Java et sous différents OS des périphériques RS-232. Les fonctions et classes Java standardisées pour l'accès aux périphériques RS-232 sont décrites sur http://docs.oracle.com/cd/E17802_01/products/products/javacomm/reference/api/javax/comm/package-summary.html , et sont bien fournies par **PureJavaComm**.

Si votre ordinateur n'a pas de version d'**Eclipse** correcte, utiliser cette version : http://www.ensta-bretagne.fr/lebars/Share/eclipse.zip (à extraire dans **C:\Temp**).

Un code à trous avec des indications est fourni (**RazorAHRSPureJavaCommTest_students.zip** pour le cas de la Razor, s'en inspirer aussi pour les autres cas) pour pouvoir écrire un programme capable de faire à peu près la même chose que le code fourni précédemment : récupération de yaw, pitch, roll pour la Razor, récupération de la latitude et longitude pour un GPS, commande de servomoteurs pour la Lynxmotion SSC-32/SSC-32u ou Polulu Maestro. Pour l'utiliser dans **Eclipse**, faire **File\Import\Existing Projects into Workspace**. Si nécessaire, aller dans les propriétés du projet **Properties\Java Build Path\Libraries** et cliquer sur **Add JARs** et choisir **lib/purejavacomm.jar** et **lib/jna-3.5.1.jar**. Examiner ensuite le code fourni et le compléter pour qu'il fonctionne.

C/C++

Des infos potentiellement utiles sont sur https://www.ensta-bretagne.fr/lebars/tutorials/Serial%20port%20C.pdf .

Pour le cas de l'ArduPilot, le protocole **MAVLink** est utilisé pour la communication entre l'autopilote et Mission Planner ou tout autre application. En C/C++, il faudra utiliser

https://github.com/mavlink/c_library (#include "common/mavlink.h"...) dont la documentation est sur https://mavlink.io/en/messages/common.html. Les types/fonctions/constantes utiles seront notamment :

- mavlink_message_t
- mavlink_status_t
- mavlink_attitude_t
- mavlink_gps_raw_int_t
- mavlink_rc_channels_override_t
- mavlink_manual_control_t
- MAVLINK_MSG_ID_ATTITUDE
- MAVLINK_MSG_ID_GPS_RAW_INT
- mavlink_parse_char()
- mavlink_msg_attitude_decode()
- mavlink_msg_gps_raw_int_decode()
- mavlink_msg_rc_channels_override_encode()
- mavlink_msg_manual_control_encode()

Il est possible que par défaut, les messages ne soient pas envoyés automatiquement, il faut donc les activer, voir par exemple :

- mavlink_msg_command_long_encode()
- MAV_CMD_SET_MESSAGE_INTERVAL
- mavlink_msg_request_data_stream_encode()
- MAV_DATA_STREAM_RAW_SENSORS

Des paramètres internes peuvent aussi devoir être changés, voir par exemple :

- mavlink_msg_param_set_encode()
- ARMING_CHECK
- SYSID_MYGCS (might be removed from recent versions) or **MAV_GCS_SYSID** and MAV_GCS_SYSID_HI : there is a concept of system_id and component_id in the MAVLink protocol so these parameters help to choose which software (which should have its specific system_id, component_id) has control on the autopilot.

Pour le contrôle plus ou moins direct de servomoteurs ou ESCs de moteurs, 3 méthodes sont possibles :

- **RC_CHANNELS_OVERRIDE** : message pour remplacer les signaux du récepteur connecté éventuel. C'est sans doute la méthode qui permet le contrôle le plus direct des actionneurs, bien qu'en fait ce ne soit pas forcément directement les commandes qui leurs sont envoyées (dépend notamment du mode dans lequel est l'autopilote, des paramètres SERVO1_FUNCTION, SERVO3_FUNCTION, PILOT_STEER_TYPE, PIVOT_TURN_ANGLE, etc.). Semble non supporté pour ArduRover < V4.0.0 (OK pour SITL V4.0.0) et non supporté pour ArduSub (dernière version testée : version préliminaire V4.2.0).
- **MANUAL_CONTROL** : message fait à l'origine pour contenir les commandes venant d'un joystick (mais il se peut que via Mission Planner, le contrôle via joystick utilise RC_CHANNELS_OVERRIDE…). Selon le mode dans lequel est l'autopilote, on aura un contrôle plus ou moins direct (au mieux non linéaire ?) similaire à RC_CHANNELS_OVERRIDE. Semble non supporté pour ArduRover < V4.0.0 (OK pour SITL V4.0.0).

- **SET_ATTITUDE_TARGET** : messages pour l'envoi de consignes de stabilisation en attitude, nécessite généralement d'être en mode Guided ou Guided_NoGPS. Voir aussi http://ardupilot.org/dev/docs/copter-commands-in-guided-mode.html. Pas vraiment adapté pour ArduSub du fait que l'attitude du BlueROV ne soit pas liée à ses mouvements en translation (dernière version testée : V4.0.3).

Il est aussi possible de voir les messages MAVLink échangés avec Mission Planner dans **CONFIG\Planner\Layout\Testing screen\MAVLink Inspector**.

Du code existant potentiellement utile est disponible sur https://github.com/mavlink/mavlink/blob/fd7b31ca4b856b66081bca5634aff5a263a81ede/examples/linux/mavlink_udp.c, https://github.com/ENSTABretagneRobotics/UxVCtrl/blob/master/Hardware/MAVLinkDevice.h (those 2 code samples are designed to connect to an autopilot running ArduPilot firmware family), https://www.ensta-bretagne.fr/lebars/Share/fake_autopilot.zip, https://github.com/ENSTABretagneRobotics/UxVCtrl/blob/master/MAVLinkInterface.cpp (those 2 code samples act as a sort of autopilot simulator and it is assumed Mission Planner is the GCS used to communicate with them). Les exemples de la partie Python sont aussi utiles pour comprendre la logique de certains messages.