Projet
conception d'un robot mobile

Professeur : Valentin Gies

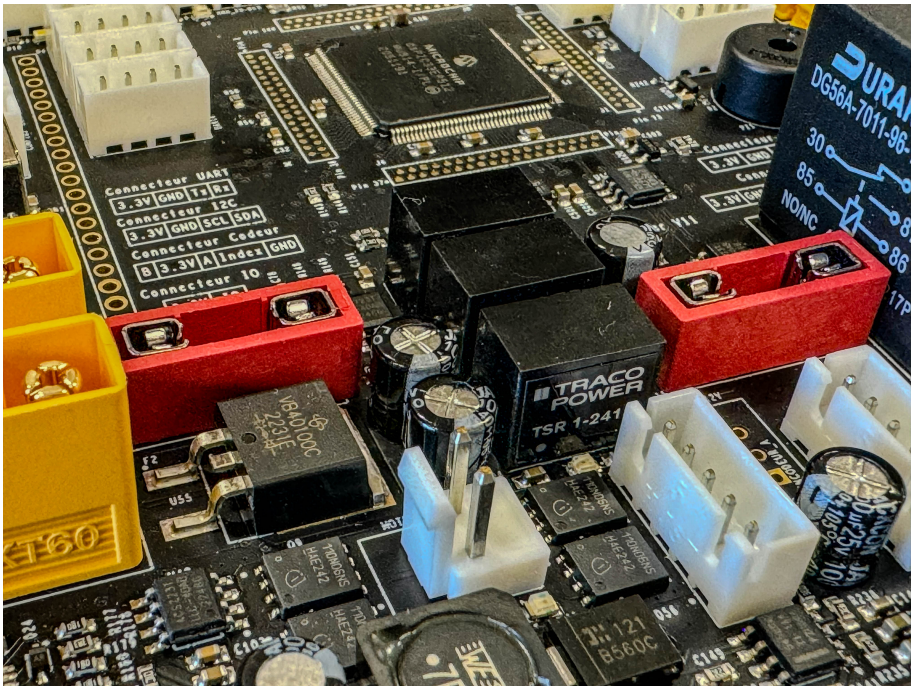

Table des matières

1 En route vers la programmation évènementielle

Après avoir programmé la partie embarquée de votre robot, une interface graphique de supervision et pilotage, la liaison permettant d'échanger des messages entre l'embarqué et le PC, la gestion de l'odométrie et de l'asservissement de vitesse en embarqué, vous avez découvert dans ces deux grandes premières parties ce que l'on appelle la programmation de bas niveau (proche des composants, en C) et la programmation de moyen niveau (en C#). Comme vous devez commencer à être à l'aise dans ces deux langages, nous allons passer à une troisième partie : la programmation évènementielle de haut niveau.

Comme vous avez pu le ressentir dans les parties précédentes, sans une structuration claire et parfaitement respectée, le code devient rapidement confus, surtout quand on ne sait pas parfaitement où placer de manière pertinente les ajouts. Le besoin d'une structuration forte a donc du vous paraître une priorité, l'objectif étant si possible d'aller vers un projet dans lequel plusieurs personnes doivent travailler sur le même code au même moment. Cela suppose un **découpage du projet en modules indépendants** et d'utiliser des **outils de développement collaboratifs**.

1.1 Développement modulaire basé sur programmation évènementielle

Dans cette partie nous allons utiliser les bases robotiques avancées destinées à participer à la compétition Eurobot. Elles sont constituées de :

- une carte embarquée intégrant un dsPIC33EP512MU814, 7 contrôleurs moteurs d'une puissance unitaire de 1kW en pointe, de 10 encodeurs quadratiques permettant de gérer l'odométrie, 3 liaisons série, une liaison USB native, 4 SPI et 3 I2C. Il est à noter que la liaison USB sera utilisée sur cette carte au lieu de la liaison série utilisée précédemment, de manière à augmenter le débit pouvant être transmis entre la carte moteur et le PC.
- un PC embarqué de type LattePanda Alpha 864, avec 8GB de RAM, et un processeur INTEL M3.
- Deux moteurs Maxon DCX26 avec un réducteur ayant un rapport de réduction de 19.2.
- Un LIDAR TIM561 de chez SICK permettant de cartographier dans le demi-plan horizontal avant l'environnement dans un rayon de 10m.

L'objectif de cette partie est d'arriver à faire fonctionner ensemble les différentes parties de haut niveau du robot décrites à la Figure **??**, et de pouvoir contribuer à plusieurs d'entre elles, en particulier :

- Positionnement du robot
- Perception de l'environnement à l'aide du lidar Sick TIM561.
- Génération de trajectoire
- Gestion des stratégies

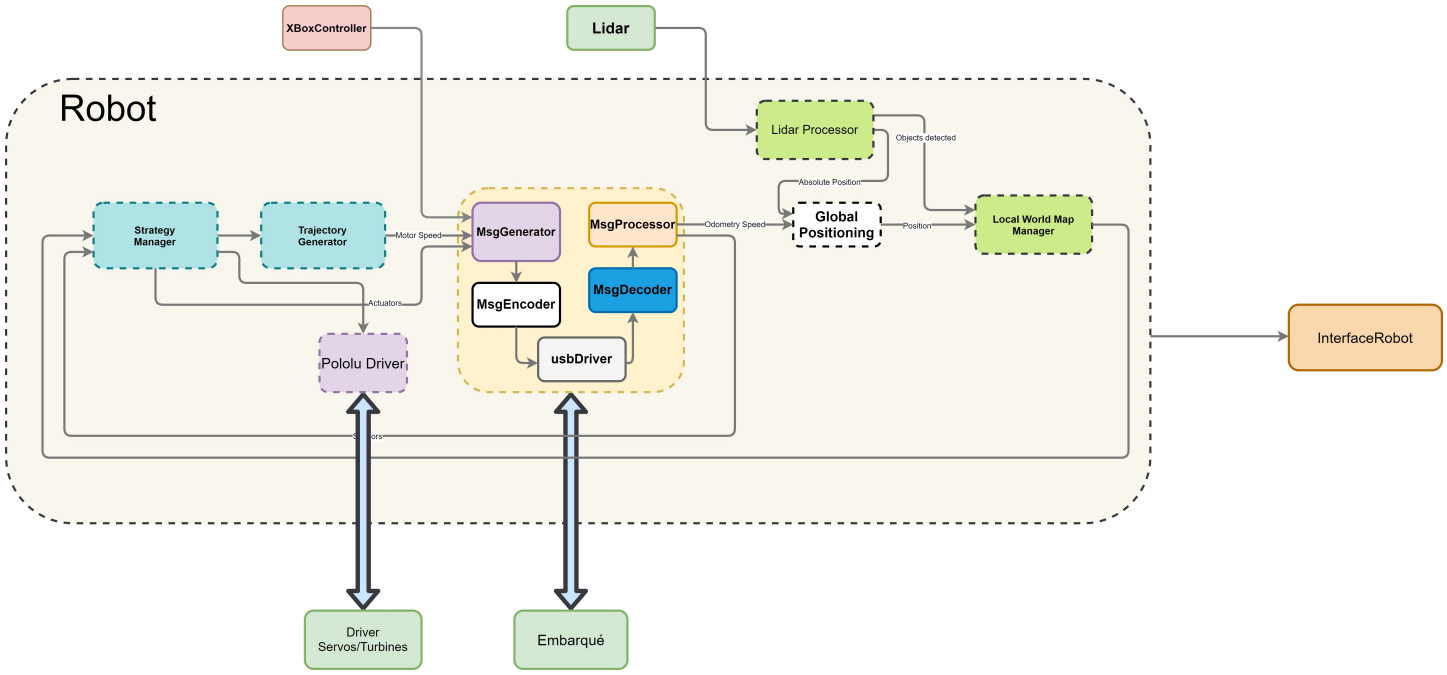

FIGURE 1 – Architecture du robot

⟹ Pour commencer, vous allez faire un *fork* de l'ossature logicielle du robot décrite à la Figure **??**. Pour cela, en étant connecté en ligne depuis votre compte Github (pas depuis GitHub Desktop...), allez sur le projet https://github.com/iutgeiitoulon/EurobotStructure. Cliquez sur Fork en haut à droite, cela va créer une copie de ce repo dans votre répertoire, qui restera liée à la version source afin de propager d'éventuelles modifications ultérieurement.

⟹ Une fois le *fork* réalisé, changer le nom de votre copie de repo en utilisant la procédure décrite avant et partagez le avec le compte *iutgeiitoulonE105*. Demandez au professeur de valider le partage (il a du recevoir un email de confirmation).

Vous avez à présent une version d'un code *C#* implantant l'ossature logicielle du robot décrite à la Figure **??**, mais dont certains blocs ne sont pas remplis ou terminés.

⟹ Ouvrez et compilez la solution *RobotEurobot2Roues.sln* présente dans le répertoire :
C :\Github\EurobotStructure\RobotEurobot2Roues. Si elle ne compile pas demandez de l'aide au professeur.

⟹ Prenez à présent le temps d'analyser et de comprendre le fonctionnement de l'architecture de ce projet complexe. En particulier, vous pouvez constater qu'il est formé de nombreux projets indépendants (modules déclarés en *static*) ayant chacun des évènements d'entrée et des évènements de sortie (à voir à l'intérieur des modules), et que ces modules sont inter-connectés entre eux tel un diagramme Simulink (mais de manière textuelle) dans le fichier *RobotEurobot2Roues.cs*. Vous noterez également que l'application de base de la solution est une application de type *Console* (et non graphique), et que l'interface graphique est instanciée dans un Thread depuis cette application Console (dans la fonction *static void StartRobotInterface()*). Les connexions des modules vers l'interface graphiques sont d'ailleurs déclarés dans un second temps, une fois que cette interface est créée, et qu'elle s'est ouverte (cf. la fonction *static void RegisterRobotInterfaceEvents(object sender, EventArgs e)*).

1.2 Positionnement du robot

Nous allons à présent remplir le module de positionnement dénommé *Positionning2Wheels*. Celui est déclaré dans le code que vous avez forké, mais il n'est pas lié aux modules d'affichage de la position.

⟹ Vous devez donc :
- Vérifier à l'aide d'un point d'arrêt (F9) que la fonction *OnOdometryRobotSpeedReceived* du module *Positioning2Wheels* est bien appelée.
- Déclencher l'envoi d'un event de transmission de la position calculée dans le module *Positioning2Wheels*.
- Connecter l'évènement de sortie *OnCalculatedLocationEvent* de ce module *Positioning2Wheels* à l'évènement d'entrée *OnPhysicalPositionReceived* du *localWorldMapManager* à l'aide de ligne de code suivante : *positioning2Wheels.OnCalculatedLocationEvent += localWorldMapManager.OnPhysicalPositionReceived ;*
- Vérifier que l'évènement *OnPhysicalPositionReceived* du *localWorldMapManager* est bien déclenché en chaine sur l'arrivée de données odométriques en provenance de la carte moteur.
- Connecter l'évènement de sortie *OnLocalWorldMapForDisplayOnlyEvent* du *localWorldMapManager* avec l'évènement d'entrée *OnLocalWorldMapStrategyEvent* de *interfaceRobot*.
- Valider l'affichage du robot dans l'interface graphique (il ne bouge pas pour l'instant).

⟹ Une fois toutes ces opérations de tuyauterie effectuées, vous pouvez passer au code permettant de mettre à jour la position du robot en fonction des données de vitesse polaire reçues depuis la carte moteur. Il est à noter que *Vx* est la vitesse linéaire du robot, et que *Vy* est toujours nulle. Validez avec le professeur le bon fonctionnement de ce module et son affichage.

1.3 Perception de l'environnement à l'aide du lidar SICK TIM561

1.3.1 Intégration du TIM561

Dans cette partie vous allez intégrer le driver du Lidar Sick TIM561 au robot de manière à visualiser les points lidar sur les cartes du robot et le signal lidar sur l'oscilloscope dédié.

⟹ Tout d'abord regardez si le logiciel SOPAS est présent sur le PC (il est nécessaire pour avoir les drivers du TIM561). Si il n'est pas présent installez le sur le PC en le téléchargeant au préalable depuis le site de chez SICK.

⟹ Connectez votre lidar à l'aide d'un cordon micro USB (**ATTENTION : la connectique est fragile**) relié à votre PC. Ouvrez SOPAS sur le PC, vous devriez voir apparaitre le lidar si il est correctement alimenté (via une alim de labo sous au moins 15V). Connectez vous depuis SOPAS au TIM561 en double-cliquant dessus dans l'interface. Dans l'onglet *monitor* → *Scan view pro*, vous pouvez avoir accès à la visualisation des données Lidar en temps réel. Dans l'onglet *Service* → *Operating Data*, vous pouvez obtenir le numéro de série du lidar qui vous servira ultérieurement, pensez à le noter.

⟹ Ajoutez le projet Lidar présent dans le répertoire *C :\Github\EurobotStructure\LidarTIM561* à la solution (*Add Existing* dans la solution), ajoutez une référence du projet *LidarTIM561* au projet *RobotEurobot2Roues* et compilez la solution pour vérifier qu'il n'y a pas d'erreurs.

⟹ A présent, déclarez dans *RobotEurobot2Roues* une instance *static* de la classe TIM561 dénommée *lidar*. Initialisez là dans le *Main* en spécifiant en paramètres l'id du robot, le numéro de série du lidar, l'angle minimum de visualisation en radians, l'angle maximum de visualisation en radians et le flag *IsUpsideDown* à true car le lidar à la tête à l'envers sur les robots.

⟹ Afin de lancer le lidar, il faudra également ajouter un *lidar.Start()* judicieusement dans le programme.

⟹ Le lidar est à présent en fonctionnement, reste à transmettre l'event lidar OnLidarDecodedFrameEvent au *localWorldMapManager* pour afficher les points lidar sur la carte, et à l'interface pour afficher l'oscilloscope lidar. Les events d'entrée existent déjà, à vous de les trouver dans le code et à les lier.

Perception de l'environnement grâce au lidar

$\Longrightarrow$ Normalement, vous devriez être en mesure de visualiser les données lidar sur l'interface dans les cartes et sur l'oscilloscope dédié. Validez avec le professeur.

$\Longrightarrow$ Toutefois, aucun code de post-processing lidar n'est encore implanté, mais vous allez le faire par vous même dans un module *LidarProcessor* à créer et à raccorder avec des events à l'application, avec deux objectifs :

- Trouver les bordures horizontales du terrain Eurobot. Pour cela il serait intéressant de rechercher dans la littérature scientifique commet trouver une ligne droite dans un ensemble de points lidar. De bonnes sources peuvent se trouver sur IEEE Xplore (accès payant) ou sur MDPI (open access). Cette tâche n'est pas simple du tout. Demandez au professeur de vous conseiller sur le choix des algorithmes à implanter parmi ceux que vous aurez trouvés.
- Détecter les objets caractéristiques du règlement de l'année en cours. L'analyse des discontinuités dans le signal lidar devrait vous aider à trouver les objets saillant. Leur largeur pouvant être calculée en multipliant leur taille angulaire par leur distance sera d'un grand secours pour confirmer la nature de ces objets.

1.4 Implantation de stratégies

Dans cette partie, vous allez apprendre à implanter des stratégie simples permettant de gérer le robot ou une fonction particulière de celui-ci. Nous allons pour cela commencer avec un déplacement sur un carré de coordonnées (0, -0.5), (0.5, 0), (0, 0.5) et (-0.5, 0). Le robot doit être placé initialement en (0, 0) avec un angle égal aussi à 0.

$\Longrightarrow$ Commencez par essayer de comprendre comment fonctionne le module *StrategyManagerProjetEtudiant*. Si vous avez besoin d'explications, demandez au professeur.

$\Longrightarrow$ La tâche *TaskDemoMove* va servir à envoyer des ordres au trajectory manager. Ces ordres doivent être transmis via des events lancés depuis *StrategyGenerique*. Implantez la tuyauterie d'event permettant d'envoyer un waypoint au *TrajectoryManager* depuis *StrategyGenerique*. L'évènement déclenché dans le *TrajectoryManager* existe déjà, à vous de le chercher.

$\Longrightarrow$ Envoyez les events de transmission de waypoint depuis la tâche et validez les déplacements du robot sur la table avec le professeur.