Projet final

BRAS ROBOTISÉ avec CamÉra

Samuel Duclos | 247-616 Élaborer l'installation et l'intégration d'un système ordiné | 20/9/22

Table des matières

[Table des matières 2](#_Toc68078987)

[**Description du projet** 2](#_Toc68078988)

[**Déroulement de la présentation de fin de projet** 2](#_Toc68078989)

[**Requis systèmes** 3](#_Toc68078990)

[**Requis matériels** 4](#_Toc68078991)

[**Requis logiciels** 6](#_Toc68078992)

[**Architecture système** 7](#_Toc68078993)

[**Architecture matérielle** 8](#_Toc68078994)

[**Architecture logicielle** 8](#_Toc68078995)

[**Identification et gestion des risques** 10](#_Toc68078996)

[**Planification** 11](#_Toc68078997)

[**Références** 13](#_Toc68078998)

[Liste des figures 14](#_Toc68078999)

[Liste des tableaux 15](#_Toc68079000)

Description du projet

Le projet sert à démontrer les compétences acquises durant la technique ainsi qu’à en développer de nouvelles. Il s’agit d’un système embarqué contrôlant un bras robotisé avec la rétroaction d’une caméra montée au bout afin de contrôler une suce qui attrape des objets. Le bras étant initialement dépourvu de capteur visuel, il est supposé que sa capacité à d’atteindre des objets d’atteindre des objets soit limitée dans un environnement changeant ; le but est d’y remédier. En bonus, la capacité à cibler, attraper et déplacer des objets donnés arbitrairement placés, « l’auto-build », « l’auto-deploy », l’utilisation de ROS (Robot Operating System), l’utilisation de conteneurs Docker, la simulation sur Gazebo, la simulation d’apprentissage visuomoteur par renforcement, l’apprentissage visuomoteur dans le monde réel, supporter Bash sur Ubuntu Linux, Bash sur Ubuntu Linux sur WSL2 et Bash sur Brew pour MacOS.

Déroulement de la présentation de fin de projet

La présentation débutera par un gros plan du robot sur un projecteur en arrière-plan avec une description de l’architecture matérielle. Lorsque le matériel aura été présenté, l’audience se fera demander de sélectionner un objet à trier (si l’architecture logicielle finale permet de changer les catégories d’objets rapidement). Le bras se fera présenter une série d’objets et devra en attraper certains mais pas d’autres. Une description de l’architecture logicielle s’ensuivra.

Requis systèmes

L’analyse de la description du projet traduit les exigences. En reflétant les compétences acquises dans la technique et peaufinées en recherche telles la conception et le montage de circuits imprimés, la planification, la réutilisation d’éléments pour simplifier la conception, la communication, la définition de la solution à entreprendre pour adresser un problème donné à résoudre et faciliter la planification. Le rapport doit être entamé d’avance et être mis à jour régulièrement afin d’assurer la bonne planification et la cohérence avec le projet fini. Le bras robotisé devra être muni d’une caméra. Un circuit imprimé connectera toutes les composantes électroniques du circuit, sauf la caméra qui est déjà montée sur un circuit optimisé pour la capture d’images. Ce système devra accomplir deux fonctions au minimum : contrôler le uARM en variant le PWM des servomoteurs et détecter des objets donnés arbitrairement placés à l’aide d’une caméra. La capacité à cibler, attraper et déplacer des objets donnés arbitrairement placés seront en bonus pour prioriser certains éléments et assurer une marge de manœuvre dans l’exécution de la planification du projet. Aussi en bonus, seront « l’auto-build » et « l’auto-deploy », l’utilisation de ROS (Robot Operating System), l’utilisation de conteneurs Docker, la simulation sur Gazebo, la simulation d’apprentissage visuomoteur par renforcement, l’apprentissage visuomoteur dans le monde réel, supporter Bash sur Ubuntu Linux, Bash sur Ubuntu Linux sur WSL2 et Bash sur Brew pour MacOS.

Les requis systèmes qui découlent de la traduction des exigences mentionnées précédemment figurent dans le Tableau 1

|  |  |  |
| --- | --- | --- |
| Numéro | Nom | Description |
| 1 | Compétences en TSO | Le projet reflète les compétences acquises en Technologie de Systèmes Ordinés. |
| 2 | Compétences en recherche | Le projet reflète les compétences acquises durant le projet. |
| 3 | Bras robotisé et caméra | Le projet comporte un bras robotisé muni d’une caméra |
| 4 | Circuit imprimé | Le circuit imprimé du bras est remplacé et amélioré |
| 5 | Contrôle des moteurs | Le bras bouge selon des coordonnées |
| 6 | Objets | Le bras détecte et attrape des objets arbitrairement placés |
| 7 | La précision du bras (bonus) | La précision du bras en utilisant une caméra et de électronique plus précis est améliorée pour attraper des objets |

Tableau 1: Identification et description des requis systèmes

Requis matériels

La définition du matériel est simplifiée pour permettre la mise à jour lors de l’itération visant à établir le matériel à utiliser de manière plus précise. Le *uARM* est choisi pour son accessibilité au magasin de génie électrique, pour limiter les coûts des pièces à acheter et pour réduire le temps de développement. L’électronique sera entièrement refait sur un circuit imprimé pour augmenter les capacités de calcul, de réception du capteur et de contrôle du bras. Un ESP32 sera utilisé comme élément central du centre de commandes. Il est supposé qu’une caméra montée sur le bras permettra de développer une boucle de rétroaction plus précise et versatile qu’avec le capteur de proximité du bras original. Un connecteur MiniUSB-A et un circuit intégré FTDI servira à la liaison USB/série permettant la programmation du microcontrôleur. Trois servomoteurs seront contrôlés par PWM et un convertisseur analogue à digital servira à lire l’angle des flexions du bras. Un Nvidia Jetson servira à programmer le bras et connectera un Arducam Camarray à deux caméras haute-vitesse. Trois connecteurs serviront à brancher les servomoteurs et un autre connecteur servira à brancher la pompe. Un interrupteur de fin de course servira pour déterminer la fin de succion de la pompe. Un circuit d’alimentation est prévu pour protéger les composantes. La caméra communiquera par MIPI-CSI pour accélérer le flux vidéo.

Les requis matériels qui découlent des requis systèmes mentionnés précédemment figurent dans le Tableau 2: Identification et description des requis matériel

|  |  |  |
| --- | --- | --- |
| Numéro | Nom | Description |
| 1 | ESP32-D0WDQ6 | Circuit intégré principal |
| 2 | MiniUSB-A (3-1734035-2) | Connecteur USB pour programmation |
| 3 | 3 servomoteurs | ADC pour lire l’angle  Sortie PWM pour contrôler les moteurs |
| 4 | FTDI (FT232RL) | Conversion USB/série |
| 5 | DC Jack (DCJ235-05-A-XX-X\_REVA) | Alimentation 3.3V/5V |
| 6 | Connecteurs (215297-4) | 3 connecteurs pour les servomoteurs  1 connecteur pour la pompe  1 connecteur pour le capteur |
| 7 | Interrupteur de fin de course | Fin de succion |
| 8 | ESP32-CAM | Contient OV2640DS |
| 9 | Diodes électroluminescentes | 3 DELs pour le débogage et l’indication de l’état du système |
| 10 | PSRAM | 8MB, SPI |
| 11 | Régulateur (LM3940IT-3.3) | Alimentation 3.3V/5V |
| 12 | 1 MP OV9281 (x2) | Modules de caméra avec obturateur global et lentilles M12 à faible distortion |
| 13 | Arducam Camarray | HAT caméra stéréo |
| 14 | Câbles de caméra | 150mm/6inch 22pin câbles de caméra |
| 15 | Câbles de caméra | 300mm/12inch 22pin câbles de caméra |
| 16 | Câbles de caméra | 73mm/2.87inch 22pin câbles de caméra |
| 17 | M2.5\*10mm nylon hex standoffs (x4) | M2.5\*10mm nylon hex standoffs |
| 18 | M2.5 nylon nuts (x4) | M2.5 nylon nuts |
| 19 | M2.5\*6mm écrous en nylon (x4) | M2.5\*6mm écrous en nylon |
| 20 | ABM8-40.000MHZ-10-1-U-T | Oscillateur crystal (40MHz) |

Tableau 2: Identification et description des requis matériel

La spécification des servomoteurs commandés par modulation de largeur d’impulsion figure dans le Tableau 3: Encodage PWM pour servomoteurs. L’étendue du mouvement du servomoteur de gauche oscille entre 0° et 150°. Le servomoteur droit oscille entre 20° et 150°.

|  |  |
| --- | --- |
| Signal commande | Modulation de largeur d’impulsion |
| Fréquence de pulsation | 50Hz |
| Étendue de la largeur de la pulsation | 500 à 2500 microsecondes |
| Position neutre | 1500 microsecondes |
| Degré de course | 180°±3° (quand entre 500 et 2500 microsecondes) |
| Position neutre | 1500 microsecondes |
| Largeur de la zone morte | 2 microsecondes |
| Direction de la rotation | Sens horaire (quand entre 500 à 2500 microsecondes) |

Tableau 3: Encodage PWM pour servomoteurs

Requis logiciels

Un ordinateur portatif (x86\_64) exécute le programme principal par une connexion SSH sur le connecteur USB. Le terminal interface le système d’exploitation Linux (Ubuntu 18.04.5 LTS). Celui-ci est exécuté dans un conteneur Docker permettant la reproductibilité dans une version d’Ubuntu qui est un peu vieille et que je ne voudrais pas utiliser comme système de base sur ma plateforme de développement. Docker est utilisé en tandem avec QEMU pour émuler Ubuntu 16.04 LTS sur une architecture ARM64v8, sur x86\_64 ou sur ARM64v8. ROS (Robot Operating System) Kinetic est utilisé sur Ubuntu 16.04 LTS pour faciliter le développement rapide du système. Il permet, à travers Gazebo, d’émuler le bras dans cet engin de physique. Il permet aussi de contrôler le bras, en même temps ou non que la simulation. Les caméras permettront de capturer des perspectives différentes qui serviront à l’entraînement du modèle (plan C, FERM), qui permettront de localiser les objets donnés (plan B, jetson-inference), ou qui permettront de rouler l’inférence du réseau neuronal YOLOv4 (plan A, OpenCV).

Les requis logiciels qui découlent des requis systèmes mentionnés précédemment figurent dans le Tableau 4

|  |  |  |
| --- | --- | --- |
| Numéro | Nom | Description |
| 1 | TensorFlow/Keras/PyTorch | Simplifie la définition, l’entraînement, les tests, la validation et le déploiement d’un réseau neuronal |
| 2 | OpenCV | Simplifie le développement et les tests pour déployer un système utilisant la vision machine, ici utilisé pour détecter des objets et simplifier la définition du pipeline permettant de tirer profit d’accélérations matérielles et/ou de fonctionnalités vidéo à travers Gstreamer, FFMPEG, V4L2 par exemple |
| 3 | ESP-IDF | Librairies disponibles pour accélérer le développement de prototypes pour l’Internet Connecté des Objets sur ESP |
| 4 | GCC | Utilisé pour le « (cross-)compile » d’un ESP-IDF ou d’un Jetson personnalisé |
| 5 | ROS | Facilite le développement rapide de systèmes |
| 6 | Linux (Ubuntu 18.04.5 LTS) | Facilite le développement rapide de systèmes |
| 7 | Docker | Assure la reproductibilité et le déploiement efficace du code développé (ici, le même code est utilisé sur ARM64v8 et x86\_64 et le système d’exploitation désuet qui est émulé ne contraint pas la plateforme de développement hôte) |
| 8 | QEMU | Permet l’émulation de l’architecture ARM64v8 sur x86\_64 ou sur ARM64v8. Le même code est utilisé sur ARM64v8 ou sur x86\_64 pour émuler un système d’exploitation désuet sur ARM64v8 sans contraindre la plateforme de développement hôte (qui peut être ARM64v8 ou x86\_64). |
| 9 | Gazebo | Engin de physique permettant la simulation graphique du bras et des caméras |
| 10 | OpenAI Gym | Librairie facilitant la définition d’environnements pour pratiquer l’apprentissage par renforcement |

Tableau 4: Identification et description des requis logiciels

Architecture système

La Figure 1 présente l’architecture système

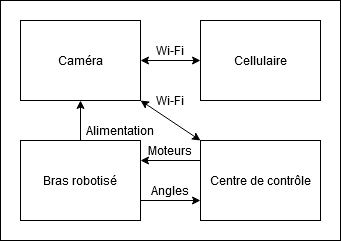


Figure 1: Architecture système

Architecture matérielle

La Figure 2 présente l’architecture matérielle

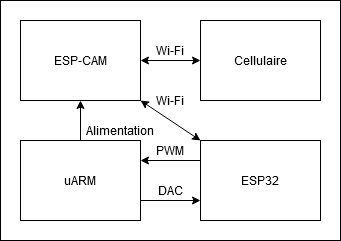


Figure 2: Architecture matérielle

Architecture logicielle

La Figure 3 présente l’architecture logicielle

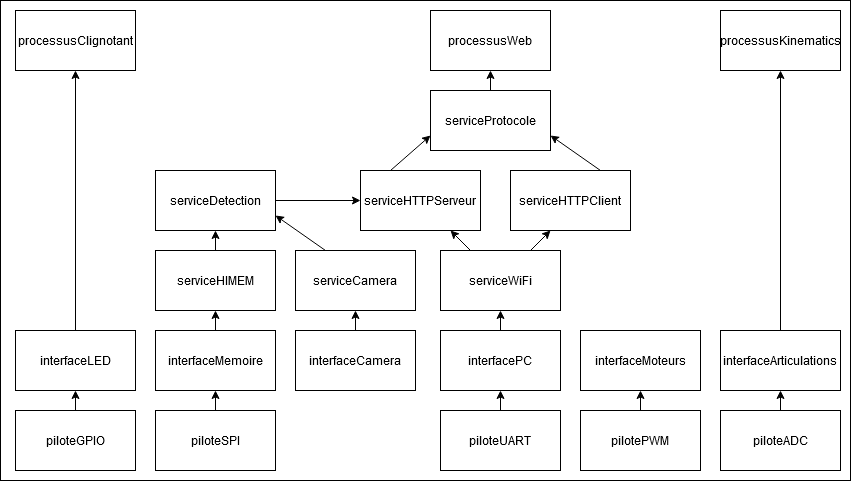


Figure 3: Architecture logicielle

La Figure 4: Architecture logicielle: réseau neuronal présente l’architecture originale MobileNet

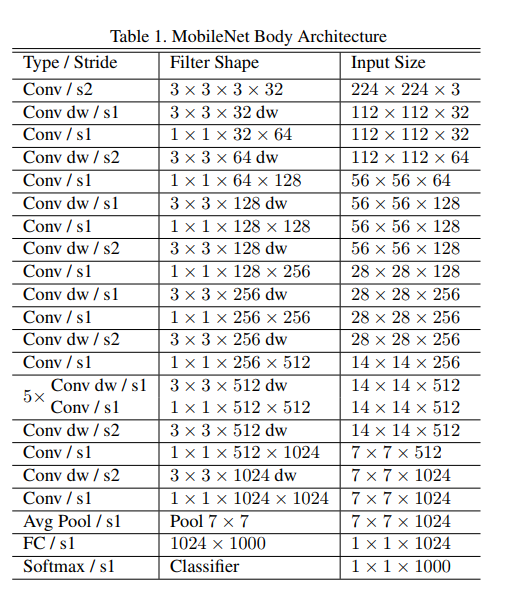


Figure 4: Architecture logicielle: réseau neuronal

Identification et gestion des risques

* Fonctionnement non complet
* Démonstration non complète
* Changement de direction
* Demander une rétroaction trop tard
* Choisir les mauvais outils
* Mauvais cas d’utilisation
* Attentes irréalistes
* Implémenter trop d’automatisation
* Bris/arrêt de fonctionnement impromptu
* Implémentation tardive de tests ou de documentation
* Maintenance ou débogage coûteux
* Mauvaise planification/gestion des risques

Planification

EN TOUT TEMPS :

* Accumuler des outils de conception de circuit imprimé haute-vitesse.
* Accumuler des outils de simulation
* Accumuler de la documentation.
* Prendre 15 minutes chaque jour pour évaluer ce qui pourrait avoir fonctionné ou pas et porter attention au déficit d’attention pour rectifier rapidement lorsque ça arrive. Noter les succès et les échecs pour le rapport et pour changer de stratégie régulièrement.
* Demander l’approbation du projet et prendre des notes des rétroactions des professeurs.
* Poser des questions à qui je peux pour maximiser l’apprentissage ultra-rapide.
* Tester tout ce qui génère des doutes (surtout ce qui est relié à la conception haute-vitesse).
* Sauvegarder tout partout en tout temps.

SEMAINE 1 :

* Écrire requis client (buts vs bonus). (fait)
* Activer VNC dans Docker sur ARM64v8 et sauver un écran, clavier et souris! (fait)
* Réparer bug de udev (fait)
* Planifier les prochaines semaines. (en cours)

SEMAINE 2 :

* Tester le container pour uARM dans Docker avec ROS sur Nvidia Jetson Nano. (fait, deadline lundi)
* Choisir/définir matériel et commander/emprunter le matériel. (fait, deadline mardi)
* Planifier les prochaines semaines avec des chemins alternatifs où il reste des risques et officialiser dans Trello. (en cours, deadline mercredi)
* Schématiser l’architecture matérielle dans DRAWIO et produire Bill Of Materials. (en cours, deadline jeudi)
* Tests de build/run en utilisant le même code sur ARM64v8 que sur x86\_64, avec émulation optionnelle en Docker avec QEMU. Ne pas dépenser plus de 3h sur ça, parce que ça fonctionne déjà sur ARM64v8. Ferait gagner du temps de build. (deadline samedi)

SEMAINE 3 :

* Schématiser l’architecture système dans DRAWIO. (deadline lundi)
* Schématiser l’architecture logicielle dans DRAWIO. (deadline lundi)
* Conception de la schématique du circuit imprimé. (deadline mardi)
* Description préliminaire des algorithmes utilisés (attendre avant de mettre trop de détails). (deadline mercredi)
* Schémas préliminaires des algorithmes utilisés. (deadline jeudi)
* Ajouter la caméra sur ROS. (deadline mercredi)
* Simuler le circuit électrique. (deadline jeudi)
* Étudier conception haute-vitesse (voir Texas Instruments application notes). (deadline vendredi)

SEMAINE 4 :

* Début du plan A (architecture logicielle) : détection d’objets avec YOLOv4. (deadline lundi)
* Officialiser en réécrivant la publication (très brouillon, simple mais détaillé pour sauver du temps plus tard). (deadline mardi)
* Empreintes terminées. (deadline mercredi)
* La réception du matériel devrait débuter. (deadline jeudi)
* Tester l’entrée de la caméra. (deadline vendredi)

SEMAINE 5 :

* Continuation du plan A.
* Début de l’installation d’OpenAI Gym sur ROS. (deadline lundi)
* Début du plan B (architecture logicielle) : installer jetson-inference sur ROS et tester. (deadline mercredi)
* Conception de la disposition du circuit imprimé. (deadline vendredi)

SEMAINE 6 :

* Continuation du plan B.
* Début de l’écriture de la définition de l’environnement OpenAI Gym. (deadline lundi)
* Complétion du plan A. (deadline mercredi)
* Les pièces devraient être toutes arrivées (deadline vendredi).

SEMAINE 7 :

* Début du plan C (architecture logicielle) : étudier la conception d’environnement sur OpenAI Gym en écrivant un crypto\_bot ou stocks\_bot simple qui vend et achète automatiquement en apprenant sur le tas par renforcement. NE PAS prendre plus de 2 jours pour perfectionner le bot. Le but est d’apprendre les conventions de la librairie; pas de faire de l’argent. (deadline lundi)
* Complétion du plan B. (deadline mardi)
* Remise des fichiers de production du circuit imprimé. (deadline mercredi)
* Commander au moins 2 circuits imprimés (avoir pièces prêtes au moins en double). (deadline mercredi)

SEMAINE 8 :

* Continuer plan C : écrire environnement OpenAI Gym pour le uARM dans le Docker ROS.
* Souder un seul circuit imprimé sur 2. (deadline vendredi)

SEMAINE 9 :

* Début de la réécriture de FERM avec l’environnement OpenAI Gym pour le uARM dans container Docker sur ROS. (deadline lundi)
* Entraîner réseaux neuronaux et tester. (deadline vendredi)

SEMAINE 10 :

* Valider les réseaux neuronaux. (deadline mercredi)
* S’assurer de la cohésion matériel/logiciel avec le plan A et B. (deadline jeudi)
* Complétion du plan C. (deadline vendredi)
* Procéder aux derniers tests matériels. (deadline samedi)

SEMAINE 11 :

* Continuation et complétion de tout.
* Débogage, réglages, perfectionnement. (no deadline)
* Perfectionner publication. (deadline vendredi)

SEMAINE 12 :

* Semaine réservée aux imprévus.
* Sauvegardes partout!!!
* Temps en extra pour finaliser et nettoyer.

Références

Code à date:

<https://github.com/abstractguy/TSO_project>

Références matérielles:

<https://grabcad.com/library/uarm-metal-1>

<http://download.ufactory.cc/docs/en/uArm-Metal-Developer-Guide.pdf>

<https://www.uctronics.com/download/cam_module/OV2640DS.pdf>

<https://dl.espressif.com/dl/schematics/esp32_devkitc_v4-sch.pdf>

<https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf>

https://github.com/realizator/stereopi-kicad-schematic/

<https://www.raspberrypi.org/documentation/hardware/camera/schematics/rpi_SCH_Camera2_2p1.pdf>

https://www.raspberrypi.org/documentation/hardware/camera/

<https://www.raspberrypi.org/documentation/hardware/camera/schematics/rpi_SCH_HQcamera_1p0.pdf>

<https://github.com/RidgeRun/NVIDIA-Jetson-IMX477-RPIV3>

<https://www.buyapi.ca/product/raspberry-pi-8mp-camera-board-v2/>

<https://www.uctronics.com/arducam-1mp-2-stereoscopic-camera-bundle-kit-for-raspberry-pi-nvidia-jetson-nano-xavier-nx-two-ov9281-global-shutter-monochrome-camera-modules-and-camarray-stereo-camera-hat.html>

https://www.espressif.com/sites/default/files/documentation/esp32\_hardware\_design\_guidelines\_en.pdf

Références logicielles:

<https://arxiv.org/abs/1704.04861>

<https://github.com/espressif/tensorflow/blob/master/tensorflow/lite/micro/examples/doorbell_camera/training_a_model.md>

<https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.py>

https://github.com/JetsonHacksNano/CSI-Camera

<https://github.com/DrYerzinia/RaspberryPiCamerav21>

<https://developer.ridgerun.com/wiki/index.php?title=Jetson_Nano/Development/Building_the_Kernel_from_Source>

http://wiki.ros.org/ROS/Tutorials

[http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29](http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber(python))

<https://www.arducam.com/docs/camera-for-jetson-nano/multiple-cameras-on-the-jetson-nano/using-arducam-camarry-on-ros/>

<https://github.com/ArduCAM/MIPI_Camera>

<https://github.com/madhephaestus/ESP32Servo>

<https://www.youtube.com/watch?v=cDbd0ASkMog>

https://github.com/ExploreEmbedded/hornbill-blue

Liste des figures

[Figure 1: Architecture système 6](#_Toc51845922)

[Figure 2: Architecture matérielle 7](#_Toc51845923)

[Figure 3: Architecture logicielle 7](#_Toc51845924)

[Figure 4: Architecture logicielle: réseau neuronal](../../../../../../../U:/Avant-derniere%20session/Yves/DOC/Projet.docx#_Toc51845925) 8

Liste des tableaux

[Tableau 1: Identification et description des requis systèmes 3](#_Toc51846192)

[Tableau 2: Identification et description des requis matériel 4](#_Toc51846193)

[Tableau 3: Encodage PWM pour servomoteurs 5](#_Toc51846194)

[Tableau 4: Identification et description des requis logiciels 6](#_Toc51846195)