Facultad de Ingeniería Universidad de Buenos Aires

75.99 - Trabajo Profesional

MLC version 3 (Machine Learning Control)

Tutor: Adriana Echeverría
Co-Tutor: Julia Garibaldi

Integrantes:

Padrón	Nombre	Email
89579	Torres Feyuk, Nicolás R. Ezequiel	ezequiel.torresfeyuk@gmail.com
86882	Germano Zbrun, Marco	marco.germano@intraway.com
88430	Lopez Skuba, Raúl	raullopez0@gmail.com

Índice

1. Introducción Teórica

El presente trabajo se propone el análisis, diseño e implementación de un conjunto de mejoras requeridas por Thomas Duriéz, investigador del laboratorio de fluidodinámica de la facultad de ingeniería, a aplicarse sobre el sistema MLC, desarrollado por él, y utilizado como herramienta en su trabajo de investigación. El siguiente capítulo fue realizado basado en la información obtenida en [1].

1.1. Control a Lazo Cerrado optimizado a través de Función de Costo

La teoría de control clásica categoriza los sistemas de control en **Open-Loop Control Systems** y **Close-Loop Control Systems** [2]. Se definen los mismos a continuación:

- **Open-Loop Control Systems:** Son aquellos sistemas en donde la salida del sistema no es comparada contra una referencia de entrada. Estos sistemas funcionan a base de calibración. Ejemplos de los mismos puede son pavas eléctricas, balanzas, etc.. Un esquema de este modelo es mostrado en la figura 1.
- **Closed-Loop Control Systems:** También llamados sistemas retroalimentados (Feedback Control Systems). Esta clase de sistema compara la salida retroalimentada contra una referencia de entrada, obteniendo así una función de error. La señal retroalimentada (feedback signal) puede ser la salida del sistema o bien una función derivada de la misma. La función de error es utilizada para especificar la actuación a ingresar en el sistema físico. Ejemplos de los mismos son el sistema de frenos ABS o la termorregulación corporal, entre otros. Un esque de este modelo es mostrado en la figura 2.

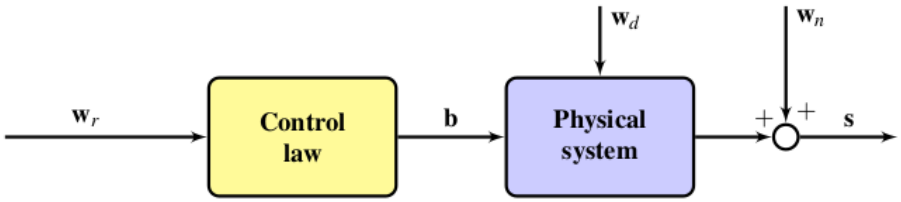

Fig. 1 Open-Loop Control System, siendo $\mathbf{w_r}$ la señal de referencia, **b** la señal de actuación, $\mathbf{w_d}$ perturbaciones externas, $\mathbf{w_n}$ ruido introducido por los sensores involucrados y **s** la señal de salida. Imagen tomada de [1]

Si bien los sistemas de control a lazo cerrado son complejos de diseñar y mantener, son los únicos que permiten estabilizar aquellos sistemas físicos que se ven afectados por estímulos externos. Esto último no es posible de realizar en un sistema de lazo abierto. Es por esta razón que este tipo de modelo es el mayormente usado para resolver problemas de naturaleza compleja (no lineales, caóticos, etc.) con un alto grado de interacción con variables externas.

Diseñar la ley de control que logre estabilizar el sistema deseado es un reto. Esto se debe a que, además de tener una comprensión matemática del sistema

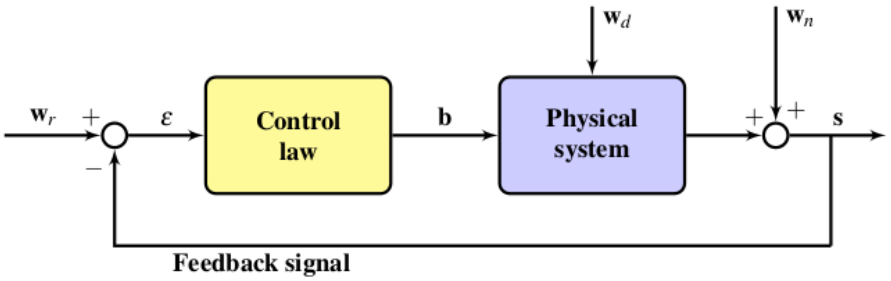

Fig. 2 Closed-Loop Control System, siendo $\mathbf{w_r}$ la señal de referencia, **b** la señal de actuación, $\boldsymbol{\varepsilon}$ la función de error, $\mathbf{w_r}$ perturbaciones externas, wn ruido introducido por los sensores involucrados y **s** la señal de salida. Imagen tomada de [1]

físico, se deben tener en cuenta todas las variables externas que modifican al mismo. Por esta razón, es importante tener una medida de la calidad de la ley de control propuesta. En la figura se muestra un sistema a lazo cerrado con la adición de una función de costo **J**. Esta función de costo es la clave para lograr definir luego un método efectivo para encontrar leyes de control a través del MLC.

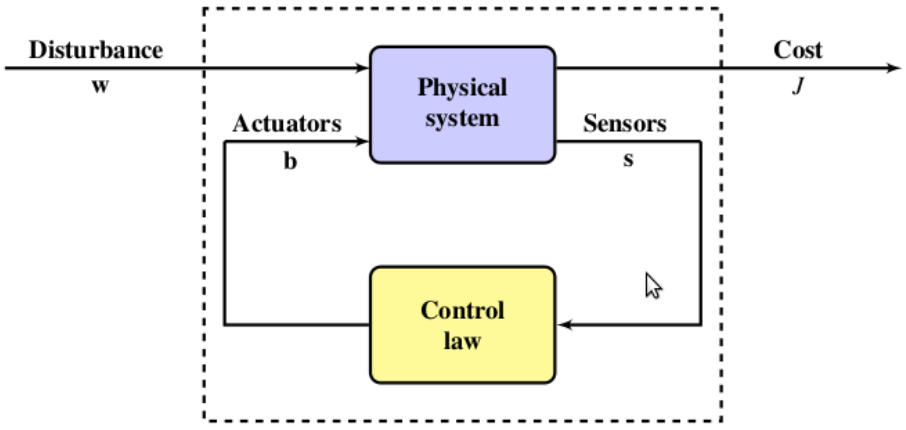

Fig. 3 Sistema de Control a Lazo Cerrado de Costo **J**. Imagen tomada de [1]

1.2. Motivación

La estabilización de flujos turbulentos es una de las ramas más estudiadas dentro del control de fluídos. Las dificultades presentes en la resolución de este tipo de problemas, sumado a la gran cantidad de campos de aplicación, hacen del mismo un tema atractivo para científicos alrededor del mundo. Se listan a continuación algunas de las características más importantes:

- La cantidad de grados de libertad (debido a la naturaleza no lineal y caótica de los mismos) **dificulta la modelización del sistema físico**

- La naturaleza no lineal de esta clase de flujos impide aplicar el principio de superposición sobre los efectos producidos por cada actuador.
- Perturbaciones externas como el *frequency crosstalk* **impiden encontrar una ley de control que lleve a la convergencia del sistema**, aún cuando se haya realizado una correcta modelización del mismo.
- Los flujos turbulentos poseen **estabilidad estacionaria** (existen valores definidos para variables estadísticas como la media y varianza). Esto permite que el sistema vuelva a su estado original luego de ser estimulado, haciendo posible la reproducibilidad de los experimentos a realizar.

Teniendo en cuenta las características enumeradas, el campo de los fluídos turbulentos permite encontrar leyes de control válidas basadas en prueba y error. Esto fue lo que motivó el desarrolló del framework MLC, el cual se describe a continuación.

1.3. MLC

1.3.1. Arquitectura

MLC es un framework utilizado para descubrir leyes de control. Tiene como fin generar modelos de sistemas de forma automática, los cuales llamaremos controladores, a través de información de la calidad de la solución generada. Esta información es procesada a través de métodos basado en *Machine Learning*, los cuales están diseñados para buscar nuevos controladores dentro del espacio de soluciones existente. La arquitectura básica del mismo se exhibe en la figura 4.

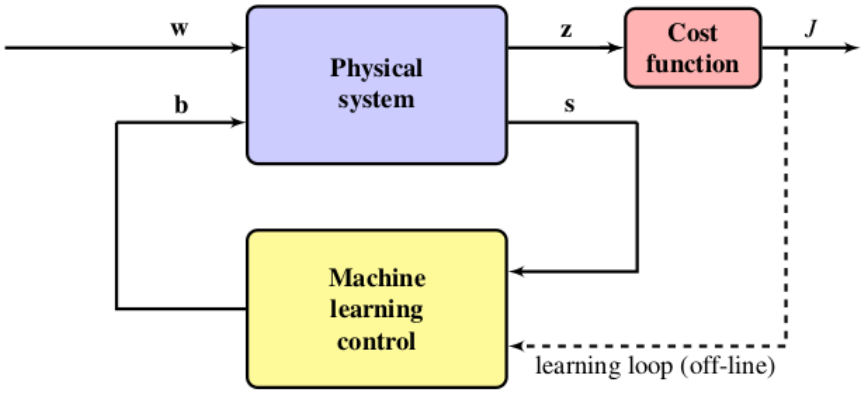

Fig. 4 Arquitectura del MLC. Imagen tomada de [1]

Comparando las figuras 3 y 4, se puede observar como la ley de control es reemplazada por el MLC, el cual recibe como entrada los valores sensados **s** y el valor **J**, resultante de la aplicación de la función de costo ante la salida del sistema físico excitado. En función de estas muestras, MLC genera una nueva ley de control la cual es evaluada, obteniendo un nuevo valor de **J**.

1.3.2. Machine Learning y Programación Genética

Machine Learning es una subrama de las ciencias de la computación basada en las campos de *Pattern Recognition* y *Computational Learning Theory*. La misma es utilizada, entre otros usos, para identificar sistemas dentro de un espacio de soluciones de alta dimensionalidad (*high dimensional search space*) a través de técnicas de aprendizaje. Existe una gran cantidad de técnicas a utilizar dentro del campo de **Machine Learning**, tales como *Gradient Search* o *Monte-Carlo Statistical Analysis*, pero las mismas poseen como característica principal la obtención de soluciones subóptimas o asociadas con un costo alto cuando el espacio de soluciones es extremadamente grande [1]. Es aquí donde entran en juego los algoritmos evolucionarios, los cuales han demostrado ser óptimos a la hora de resolver este tipo de problemas.

La *Programación Genética* y los *Algoritmos Genéticos* se basan en la propagación de generaciones de individuos, llamadas poblaciones, los cuales son seleccionados en función del valor de su *fitness*. Cada individuo tiene asociado un *costo* o *fitness* el cual es obtenido luego de evaluar el mismo a través de una función de costo. Un individuo en este ámbito *tiene una correspondencia directa con la estructura y parámetros que definen a una ley de control*. En la figura 5 se exhibe un diagrama detallado del uso de estos algoritmos dentro de la solución implementada en MLC. Un individuo $\mathbf{b} = \mathbf{K}(\mathbf{s})$ es evaluado en el sistema dinámico. De dicha evaluación, se obtiene su *fitness* $\mathbf{J}$. Al terminar de evaluar una población entera, MLC genera una nueva población a través del uso de diferentes técnicas de *Programación Genética*. Este proceso se exhibe en detalle en la sección 1.3.3.

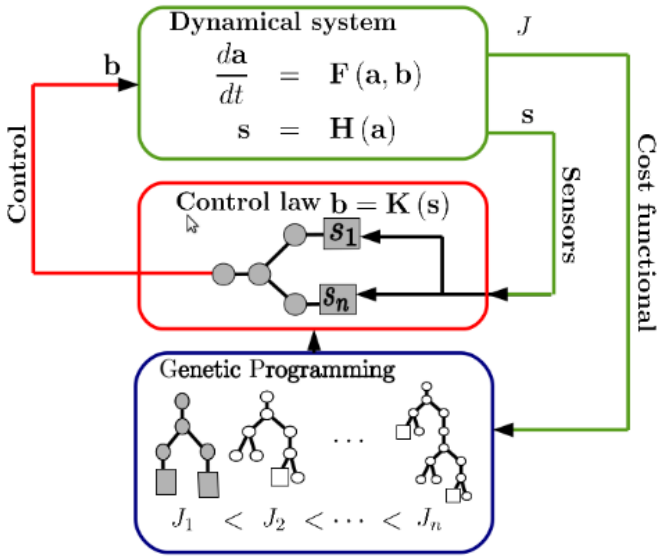

Fig. 5 Generación de nuevas leyes de control. Imagen tomada de [1]

Un individuo se encuentra modelizado dentro del MLC como un conjunto de funciones y parámetros que poseen la forma de un *recursive function tree*. Las funciones matemáticas utilizadas son las operaciones +, -, ×, / con el agregado

de cualquier otro tipo de función, como puede llegar a ser un coseno, una exponencial o una tangente hiperbólica por nombrar algunas. Los nodos hojas poseen constantes o bien parámetros del sistema (por lo general sensores), mientras que los nodos no hoja poseen las funciones matemáticas anteriormente nombradas. La figura 6 muestra la modelización de un individuo. La composición de los individuos a través de *trees* hacen de los mismos candidatos ideales para ser utilizados como input dentro de *algoritmos genéticos*.

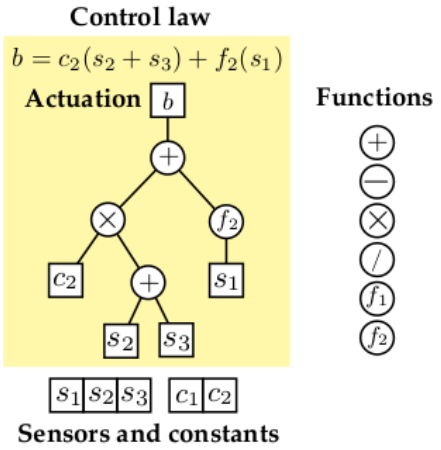

Fig. 6 Individuo $\mathbf{b} = \mathbf{K}(\mathbf{s_1}, \mathbf{s_2}, \mathbf{s_3})$ visto como un *recursive funcion tree*. Imagen tomada de [1]

1.3.3. Generación de nuevas Leyes de Control

La primera población de individuos es generada de forma aleatoria en función de los sensores, constantes y funciones matemáticas a disposición y la misma no posee un costo asociado a priori. La población luego es evaluada (el proceso de evaluación es explicado en la sección 1.3.2), con lo cual se obtiene el *fitness* asociado a cada individuo. A partir de este punto se procede a evolucionar la población. Dicha evolución se lleva a cabo a través de los siguientes algoritmos [1]:

- **Elitismo:** Se eligen a los mejores individuos de la población evaluada y se agregan directamente en la próxima generación.
- **Replicación:** De forma aleatoria, se eligen algunos individuos de la población evaluada y se hace avanzar a los mismos a la próxima población.
- **Crossover:** Se toman a dos individuos que posean un *fitness* similar y se intercambian de forma aleatoria algunas ramas del *recursive tree* asociado a cada uno.
- **Mutación:** Se modifican de forma aleatoria los parámetros o funciones de ciertos individuos.

Luego de realizar la evolución de la población se procede nuevamente a evaluar a la misma. A partir de este momento, ese proceso se repite hasta que se llega a algún punto de corte. Por lo general, el punto de corte del algoritmo utilizado por MLC consiste en la cantidad de generaciones evaluadas y evolucionadas hasta el momento. Otro punto de corte que se puede utilizar es haber alcanzado algún *fitness* deseado en el mejor individuo. En la figura 7 se muestra un diagrama de flujo explicando el proceso de generación de nuevas poblaciones.

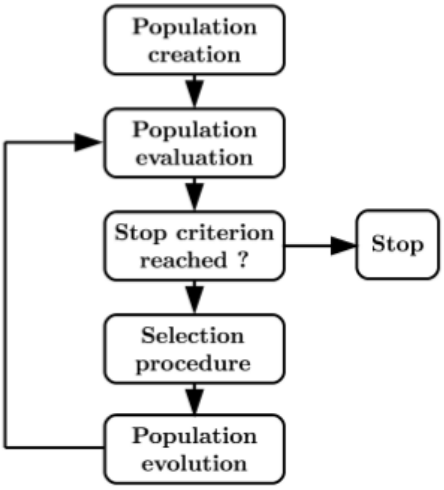

Fig. 7 Diagrama de Flujo del proceso de Generación de Poblaciones. Imagen tomada de [1]

2. Objetivo

Durante los años que Thomas ha estado trabajando en MLC se ha encontrado con una serie de dificultades técnicas y funcionales. Durante el segundo cuatrimestre del 2015, como parte de un trabajo práctico de la asignatura *75.61 Taller de Programación III* se trabajó sobre el sistema MLC, relevando requerimientos e implementando en forma parcial alguno de ellos. Debido al corto período tiempo dispuesto en la asignatura y la cantidad de tareas a realizar, se decidió continuar con el mismo como parte del presente Trabajo Profesional. Con más recursos a disposición y luego de diferentes reuniones que hemos tenido con Thomas, se llegó a una lista de objetivos en común. La misma se exhibe a continuación:

- **Migrar el sistema de MATLAB a Python:** MLC se encuentra desarrollado enteramente en MATLAB, con excepción del código que es utilizado en los dipositivos *Arduino*. Las principales razones por las cuales se desea realizar esta migración son:
  - Gracias a la cantidad de bibliotecas científicas robustas que dispone el lenguaje (*numpy, matplotlib, SciPy Stack, etc.*), Python permite desarrollar aplicaciones de procesamiento numérico y matemático de 100 % Open Source
  - Si bien MATLAB es el lenguaje más utilizado en la comunidad científica, no es un buen lenguaje para diseñar aplicaciones de software complejas. Entre las principales desventajas que posee, se pueden listar el limitado soporte a OOP (*Object Oriented Programming*) y estructuras de datos y sintaxis a medida para el procesamiento de matrices.
- **Pruebas unitarias, de integración y funcionales:** El software no posee ningún tipo de testing. Hoy en día el correcto funcionamiento del software es validado de forma manual por Thomas. Es de interés MLC es de interés agregar tests que permitan tener una medida de la cobertura de código alcanzada, como así definir métricas automáticas que permitan a futuro ponderar la calidad del sistema.
- **Refactorización de Código:** Gran cantidad del proyecto se encuentra realizado con las herramientas que MATLAB brinda a nuestra disposición. La migración a Python abre la posibilidad de cambiar el diseño actual del sistema por uno más flexible y robusto.
- **Cuellos de Botella:** Actualmente la forma en la evalúan los individuos (o su equivalente, *Control Laws*) es poco flexible. El procedimiento implica reprogramar los dispositivos *Arduino* ante cada población a ser evaluada. Esto último es llevado a cabo por temas de *performance*, debido a que ha sido imposible para Thomas evaluar cada individuo por separado. Se desea identificar y reproducir las limitaciones encontradas por Thomas y encontrar soluciones a las mismas, siempre que estas existan.
- **Interfaz Gráfica:** MLC no posee interfaz gráfica alguna. Se desea crear una interfaz de escritorio con una correcta usabilidad que permita a usuarios de la comunidad científica aprovechar al máximo las virtudes del presente framework.

3. Alcance

3.1. Requerimientos Funcionales

- Migrar el sistema completo de *MATLAB* a *Python* a través de un modo de trabajo vertical. Esto implica ir reemplazando código *MATLAB* a *Python* de forma gradual, verificando en cada paso el correcto funcionamiento del sistema.
- Utilizar el MLC desde Python con *Simulink* de forma opcional. Debido a que gran cantidad de los experimentos realizados por *Thomas* son diseñados en *Simulink*, y dado que no existe una reemplazo a esta herramienta, se debe respetar esta compatibilidad.
- Diseñar e implementar un modelo de persistencia de la aplicación basado en Base de Datos. En estos momentos los datos de las simulaciones realizadas se almacenan en formato binario. Se debe permitir poder manipular varios experimentos a la vez sin la necesidad de estar cambiando los archivos de configuración como se está realizando actualmente.
- Rediseñar el procesamiento de las leyes de control. Esto es lo que actualmente se está realizando de forma desprolija en el *Arduino*
- Diseñar e implementar un protocolo de comunicación que permita configurar y evaluar las leyes de control. Esto implica transformar los individuos de un *recursive function tree* a un formato que pueda interpretado por los dispositivos *Arduino* u otro dipositivo que respete el protocolo en cuestión. Se debe evitar, siempre que se pueda lograr, tener que quemar el *Arduino* en cada iteración.
- Desarrollar una API genérica que permita comunicar al sistema MLC con el sistema embebido. De esta forma se busca desacoplar al sistema en su totalidad de algún dispositivo de hardware específico, dejando abierta la posibilidad de poder cambiar el mismo por alguna otra plaqueta (Raspberry, beagleboard, etc.) en un futuro.
- Implementar una interfaz gráfica que permita configurar los parámetros de entrada del sistema.
- Implementar una interfaz gráfica para la visualización y el análisis de los resultados de las simulaciones realizadas.

3.2. Requerimientos No Funcionales

- El sistema debe ser multiplataforma y debe estar implementado con tecnologías de *Open Source*. Una excepción a esto es el uso de *Simulink*, como se explicó en la sección 3.1.
- Se debe garantizar que la velocidad de procesamiento de cada individuo (*controllaw*) sea menor al tiempo de actuación del sistema físico. Esto es de vital importancia a la hora de realizar el refactor sobre la evaluación de las leyes de control.

- Generar un set de pruebas de integración que permitan validar el funcionamiento del sistema con el cliente.
- Implementar un conjunto de test unitarios que garanticen la cobertura de código de los módulos del sistema de acuerdo a las métricas de calidad definidas.
- Relevar qué métricas se están usando en el laboratorio para medir la performance y qué números está manejando Thomas en este momento.
- Realizar un estudio de mercado sobre las tecnologías embebidas comerciales existentes, de forma de encontrar el producto óptimo que cumpla con la relación velocidad de comunicación / precio. Uno de los objetivos que se busca a través del MLC, es lograr que los dispositivos físicos a utilizar para el sensado de parámetros y actuación sea barato y fácil de conseguir en cualquier parte del mundo.

4. Especificaciones del Sistema

4.1. Hardware

4.1.1. MLC

MLC no posee limitaciones de hardware conocidas. Por esta razón, se toma como mínimo hardware requerido para ejecutar la aplicación el mínimo requerido por un sistema Unix. Los requerimientos mínimos se listan a continuación:

- Procesador x86 de 1 GHZ
- 512 MB RAM (1 GB recomendado)
- 20 GB de espacio en disco (la mayoría de este espacio está destinado a la instalación del MATLAB. En un futuro cuando se remueva la dependencia con MATLAB podrá reducirse este número, siempre y cuando no se utilice *Simulink*)

4.1.2. Sensado y Actuación

Los dispositivos utilizados para realizar el sensado de magnitudes físicas y la actuación de dispositivos físicos son *Arduinos*. El modelo utilizado del mismo no ha sido definido aún, pero se están realizando con pruebas con el modelo *Due*. Es posible que este modelo se cambie por otro *Arduino* o por otro dispositivo embebido que cumpla con los requerimientos necesarios.

4.2. Software

4.2.1. MLC

En su versión terminada, el sistema correrá enteramente en **Python 2.7**. Se utilizarán todas las bibliotecas matemáticas necesarias para reemplazar la funcionalidad de MATLAB en Python. A priori, se supone que con el conjunto provisto por el *Scipy Stack* debería ser suficiente, pero no se descarta tener que agregar alguna otra dependencia al proyecto.

Para realizar la migración a Python, se utilizará el módulo *Python Engine*. El mismo permite ejecutar código MATLAB en Python. Este módulo es provisto por MATLAB a partir de la versión **R2014 B**. Como se bien se ha comentado, en su versión final MATLAB no será necesario para utilizar MLC, pero existen algunas herramientas provistas por *Mathworks* como *Simulink* que no poseen un reemplazo. Por esta razón es que se agrega como una de las dependencias de software a MATLAB.

Aún no se ha decidido que framework gráfico será utilizado para realizar la interfaz gráfica de la aplicación. Se debe evaluar entre las diferentes versiones disponibles para Python (PyQT4, PyQT5, Pyside, PyGTK, wxWidgets, etc.) y elegir la que sea más conveniente.

En lo que respecta a base de datos, tampoco se ha decidido aún que motor se utilizará. Debido a que el volumen de datos manejados por experimento es baja (cota superior de 500 MB) se analizará utilizar algún motor ligero como SQLite. En caso que se necesite un motor más robusto, se analizará el uso de otras opciones como pueden llegar a ser MySQL, Postgres, MariaDB o algún tipo de motor No SQL.

4.2.2. Sensado y Actuación

El software utilizado para el desarrollo de los dispositivos embebidos está fuertemente atado al SDK liberado por cada fabricante. En el caso de *Arduino*, los mismos proveen un IDE que viene integrado con un cross-compiler compatible con el hardware disponible en cada una de las placas a disposición (*Arduino Uno, Due, Mega, etc.*). Habiendo dicho esto, todo el desarrollo con *Arduino* se realizará en C y C++.

5. Metodología

La metodología de trabajo elegida para trabajar en el proyecto es *Scrum*. El uso de esta metodología es ideal para el tipo de desarrollo a realizar, debido a que se posee un cliente real con el cual trabajar. La duración de cada uno de los sprints variará entre dos y tres semanas, dependendiendo de las tareas definidas en el sprint. En la finalización de cada sprint, se realizará una demo con el cliente mostrando los avances realizados.

Se utilizará **Trello** como herramienta de gestión de proyecto. En el mismo se encontrará el *backlog* y cada uno de los *sprints* realizados durante el transcurso del proyecto. A través de los plugins **Ollert** y **BurndownChartForTrello** se mantendrá el registro de las horas trabajadas en cada tarea, así como también se tomarán estadísticas de la eficiencia del equipo de trabajo.

El hosting del proyecto se encontrará en un repositorio privado por pedido expreso de Thomas. Al finalizar el presente trabajo se liberará el código a la comunidad. El hosting utilizado actualmente es **www.github.com**, y el mismo se utilizará además para mantener trazabilidad entre las tareas definidas en **Trello** y las modificaciones de código realizadas.

6. Cronograma

La asignatura *75.99 Trabajo Profesional* posee asociados 12 créditos en el plan de estudios de la carrera Ingeniería en Informática. Esto implica que, según el estatuto de la facultad, la misma posee una carga horaria de 24 horas semanales (12 de cursada + 12 horas de trabajo por cuenta del alumno). Teniendo en cuenta que la duración de un cuatrimestre es de 16 semanas, la cantidad de horas asignadas a cada alumno equivale a 384 horas. Este número por 3 (la cantidad de integrantes del presente proyecto), da como resultado el número total de horas del proyecto: 1152. Se redondea este número de horas a 1200 para realizar la planificación del proyecto. A continuación se define el cronograma tentativo de tareas:

Tarea	**Descripción**	**Horas**
Gestión de Proyecto	Seguimiento de tareas, análisis de métricas, planeamiento de sprints y desarrollo del proyecto	75
Reuniones de Avance	Tiempo dedicado a las reuniones a tener tanto con el cliente como con los tutores. Esto último incluye las demos a realizar en la finalización de cada *sprint*	50
Migración algoritmos genéticos	Migrar de MATLAB a Python la generación y evolución de individuos	200
Diseño e Implementación del modelo de persistencia	Se debe modelar tipos de datos a fin de persistir los experimentos en algún esquema de base de datos a definir	45
Análisis e Implementación de Gráficos	Se debe analizar que herramientas provistas por Python se amoldan a los requerimientos gráficos de la aplicación, que incluye la implementación de los gráficos existentes	40
Obtención de métricas y análisis de disp. *Arduino*	Se realizarán pruebas de desempeño de hardware de comunicación para dispositivos *Arduino*, a fin de determinar la viabilidad del procesamiento de las leyes de control en Python	30
Diseño e Implementación de Protocolo de Comunicación	El protocolo debe ser capaz de configurar los sensores y actuadores a utilizar en un experimento dado. También debe ser capaz de obtener lecturas de los sensores previamente configurados y manipular los actuadores	100
Confección de Pruebas de Integración	Se debe generar un conjunto de pruebas que permitan validar el correcto funcionamiento entre las diferentes partes que componen al MLC	30
Diseño y Armado de Entorno de Prueba Real	Se debe implementar un experimento de laboratorio que permita evaluar el correcto funcionamiento y desempeño del MLC	60

Análisis y Diseño de Mejoras de Arquitectura	Una vez que se haya migrado el MLC a Python, se procederá a evaluar mejoras en el diseño del sistema	90
Implementación de Mejoras de Arquitectura	Con la nueva arquitectura y diseño en mente, se procederá a implementar la nueva solución propuesta	270
Documentación Interna de diseño	Documentación que describa el diseño y arquitectura del sistema. La misma debe incluir diagramas clases, secuencia, etc., como así también el detalle del protocolo de comunicación implementado	60
Diseño e Implementación de Interfaz Gráfica	Se debe desarrollar una interfaz gráfica agradable para el usuario que permita la administración y configuración del sistema. La misma debe incluir configurar y visualizar los experimentos a realizar	30
Implementación de Instaladores	Se deben crear instaladores para los sistemas operativos más conocidos (Windows, MAC, Linux flavors) con el fin de facilitar el uso del MLC en la comunidad científica	60
Elaboración de Manual de Usuario	Se deben confeccionar los manuales de usuario para el uso y extensión de la versión final del sistema	60
Total Horas:		**1200**

Referencias

[1] T. Duriez and S. L. Brunton, *Machine Learning Control – Taming Nonlinear Dynamics and Turbulence*, vol. 1. Springer International Publishing, 2016.

[2] K. Ogata, *Modern Control Engineering*, vol. 17. 2002.