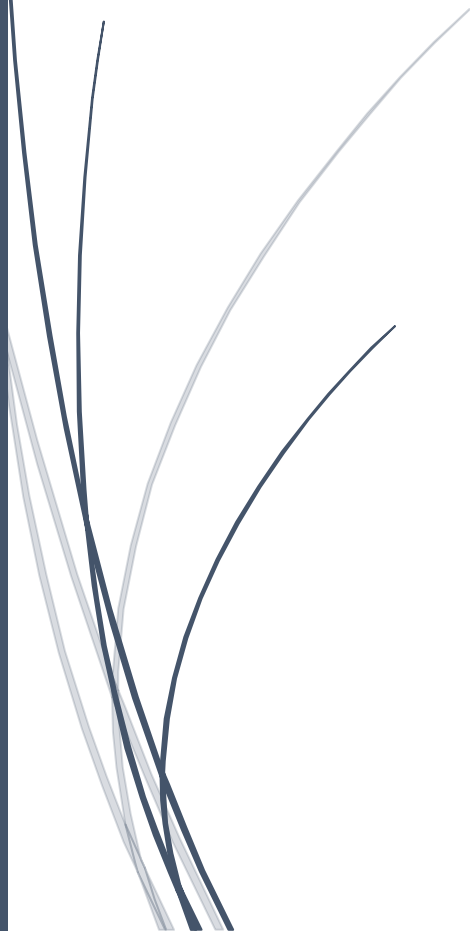
A dark blue vertical bar runs down the left side of the page. A blue arrow points to the right from this bar, containing the date.

06/26/2020

STR TP N°3

Trabajo Integrador Final

Several thin, curved lines in dark blue and light grey originate from the bottom left corner and sweep upwards and to the right.

Integrantes:
Brambilla Nicolás
Pereyra Agustín
Perez Sardi Walter

Contenido

Contenido 1

Introducción al trabajo practico..... 2

Requerimientos 2

Consignas 3

Fecha de entrega: 3

Integrantes:..... 3

Github del proyecto..... 3

Estructura del código..... 3

Manual de usuario..... 5

Mediciones..... 5

Gráficos 5

Resultado obtenido 6

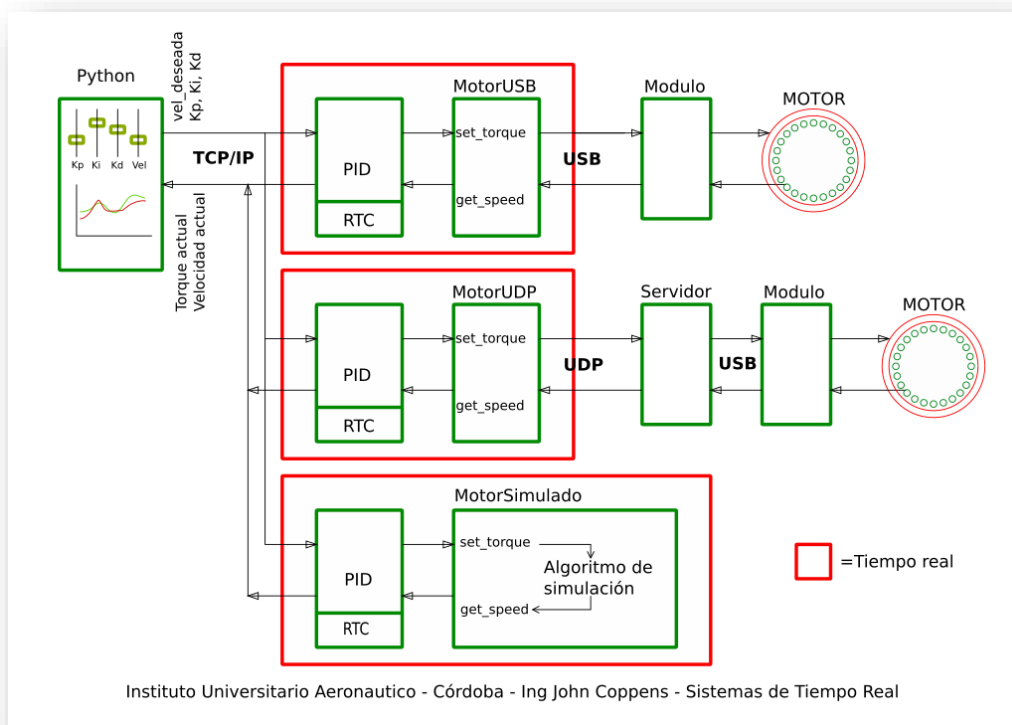
Introducción al trabajo practico

Informe del trabajo practico N°3 de la materia Sistemas en tiempo real.

La idea principal es que este trabajo practico sea el último de la materia, se incluirán todas las funcionalidades que se vienen agregando al proyecto con la finalidad de manejar el motor físico que se encuentra en el hogar del profesor John Coppens. Para eso se utilizará el protocolo UDP para enviar y recibir datos a través de un socket. Por otro lado, se dejará en funcionamiento el motor simulado.

Al principio de la materia el objetivo era llegar a utilizar el motor vía USB, lo cual no resulto por la situación que actualmente nos afecta. De todas formas, si se deseara se podría realizar ese modulo implementando la interfaz motor.h y copiando la estructura de demo_sim.c.

En resumen, la idea general es montar el proyecto como lo representa la siguiente imagen:



Requerimientos

- Respetar las reglas actuales de interacción social, i.e. es preferible colaborar por internet.
- En el examen, las presentaciones son individuales. Significa que todos los integrantes deberán estar al tanto de todas las partes del proyecto, no únicamente de su parte.
- La presentación deberá incluir un documento que sirve como 'manual de usuario', con información sobre el uso del programa, eventuales problemas, los gráficos producidos, y como repetirlos.

Consignas

- 1) El programa debe estar en una forma completa, con cualquier clase exterior (caso de Python), Makefile(s) necesarios, etc., para que se fácil de compilar si fuera necesario.
- 2) El informe debe contener, aparte de la descripción de las partes del programa, indicaciones claras de como probar el probarlo.
- 3) Tiene que funcionar con el motor simulado y con el motor 'remoto' (y preferentemente estar previsto para el módulo, aunque será difícil de hacer).
- 4) Las pruebas necesarias para realizar:
 - Con Kp, Ki, Kd constante, mostrar que el algoritmo de control es estable. Para ello, es necesario de hacer una gráfica con la velocidad deseada cambiando en pasos de 20 a 120, cada vez esperando que estabilice el motor. Ideal sería graficar también a la velocidad deseada (aparte del torque y velocidad actual que son obligatorias).
 - Con Kp, Ki, Kd constante, hacer un cambio de 0 a 120, y luego de 120 a 20.

Fecha de entrega:

Martes 23 de Junio 2020

Integrantes:

- Brambilla Nicolás
- Pereyra Agustín
- Perez Sardi Walter

Github del proyecto

https://github.com/AguPereyra/str_2020

Estructura del código

¡AVISO!: Se mantienen los archivos como base del TP N°1,2 y se agregaron los siguientes archivos que conforman el TP N°3:

motor_udp.c: Se crea el socket por donde se envía los datos.

rtc.c: Para controlar y usar el RTC (Real Time Clock) utilizado para el motor físico.

rtc.h: Inicializa el RTC de reloj con la velocidad dada y las interrupciones habilitadas.

socketserver.c: Se trabaja con sockets para abrir túneles para enviar y recibir información de la velocidad, el torque y valores como las constantes Ki, Kp y Kd.

socketserver.h: Inicializa el servicio que escuchara y comunicara a través del socket los datos.

test_udp.c: Utilizado para comprobar el envío de paquetes mediante el protocolo UDP.

utils.c: Se crea la función parse para dividir el string recibido en valores individuales (desired_speed, kp, ki, kd).

pid.c: Se calcula el torque con con kp, ki, kd y DeltaT y se lo enviamos al motor físico.

pid.h: Se establece los valores iniciales de kp, ki, kd, torque máximo y mínimo.

sim_motor.c: Se calcula el torque para el motor simulado utilizando inercia y sensibilidad.

demo_sim.c: Lee lo que entra en el socket obtiene las variables que se encuentran en el y escribe al PID la velocidad actual y el torque.

demo_udp.c: Inicializa el socket y espera por una conexión, lee lo que entra al socket obteniendo las variables y escribe al PID la velocidad actual y el torque.

Client.py: Esta clase se encarga de la comunicación a través del socket.

Connection.py: Clase que se usa como intermediaria entre la interfaz gráfica y el motor a través de un socket. Además, cada vez que recibe y envía un paquete por el socket, va a generar en un archivo llamado logs la actividad junto con la diferencia de tiempo entre la última recepción/envió de paquete y la acción actual.

logs_to_csv.py: Les da formato a los logs generados por Connection.py y los guarda en los siguientes archivos measures.csv (velocidad deseada, velocidad actual, torque, torque actual, tiempo en milisegundos) y packet_times.csv (diferencia de tiempo entre recepción y envío de paquetes). Espera como parámetro el nombre del archivo, por ejemplo, si el archivo se llama logs:

```
python3 logs_to_csv.py logs
```

main_gui.py: Encargada de crear la interfaz a cargo de manejar el motor que está conectado a través del socket en localhost:8080. Esta clase permite modificar las variables kp, ki, kd y velocidad inicial mediante línea de comando, además del nivel de registro de logs. Un ejemplo de su utilización se puede ver a continuación:

```
python3 gui/main_gui.py --log=INFO --kp=0.873 --ki=0.25 --kd=0.780 --speed=20
```

InfoLabels.py: Clase que crea los paneles que contienen los datos de la velocidad y el torque actuales para mostrarlos a través de la GUI.

SpeedObserver.py: Clase intermediaria para que la interfaz gráfica muestre la velocidad deseada.

csv_to_graph.py: Diseñado para graficar el comportamiento del motor en función de los archivos que pueden ser de dos formatos: los formatos soportados son jcoppens, def. Un ejemplo de su utilización se puede ver a continuación considerando que el archivo measures.csv contiene información del comportamiento del motor en el formato generado por el script logs_to_csv.py (formato def):

```
python3 csv_to_graph.py --from=measures.csv --format=def
```

Si se desea saber mas sobre los formatos soportados ver el script **csv_to_graph.py**

Para más información de las clases en python: Ejecutar el comando sobre la carpeta str_2020 para obtener la documentación en función de los comentarios de las mismas clases servidos en html:

```
python -m pydoc -w ./
```

Manual de usuario

Dependiendo el motor que se quiera utilizar se deberá hacer lo siguiente:

Motor UDP:

- 1) *cd real_time*
- 2) *make demo_udp*
- 3) *sudo ./demo_udp.exe*

Motor simulado:

- 1) *cd real_time*
- 2) *make demo_sim*
- 3) *sudo ./demo_sim.exe*

Como extra si se quisiera modificar las variables *kp*, *kd*, *ki* y *speed* se podrá ingresar por parámetros de la siguiente manera:

```
python3 main_gui.py --kp=value --ki=value --kd=value --speed=value
```

Mediciones

En el caso de las mediciones para guardarlas en un archivo csv se deberá ingresar el parámetro *--log=INFO*. Quedaría de la siguiente manera:

```
python3 main_gui.py --log=INFO
```

El comando almacena en el archivo logs varios datos con respecto a la ejecución del programa, entre ellos registra los datos del motor (velocidad, torques actuales, velocidad deseada, paso de tiempo que incrementa en 50 milisegundos en cada iteración), y las variaciones de tiempo entre el envío y recepción de paquetes.

Para poder acceder a estos dos conjuntos de datos de una manera mas clara, se puede utilizar el script **logs_to_csv.py** ya descripto anteriormente para obtener los datos en csv.

Gráficos

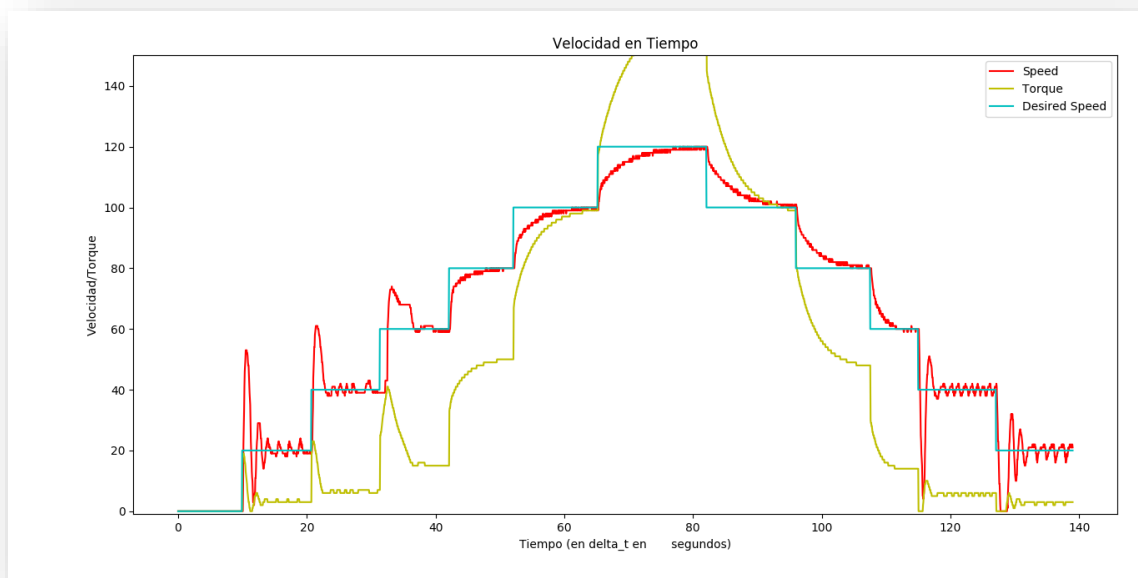
Si se desea generar gráficos con los datos contenidos en los csv se debe utilizar el script **csv_to_graph.py** ya explicado anteriormente.

Resultado obtenido

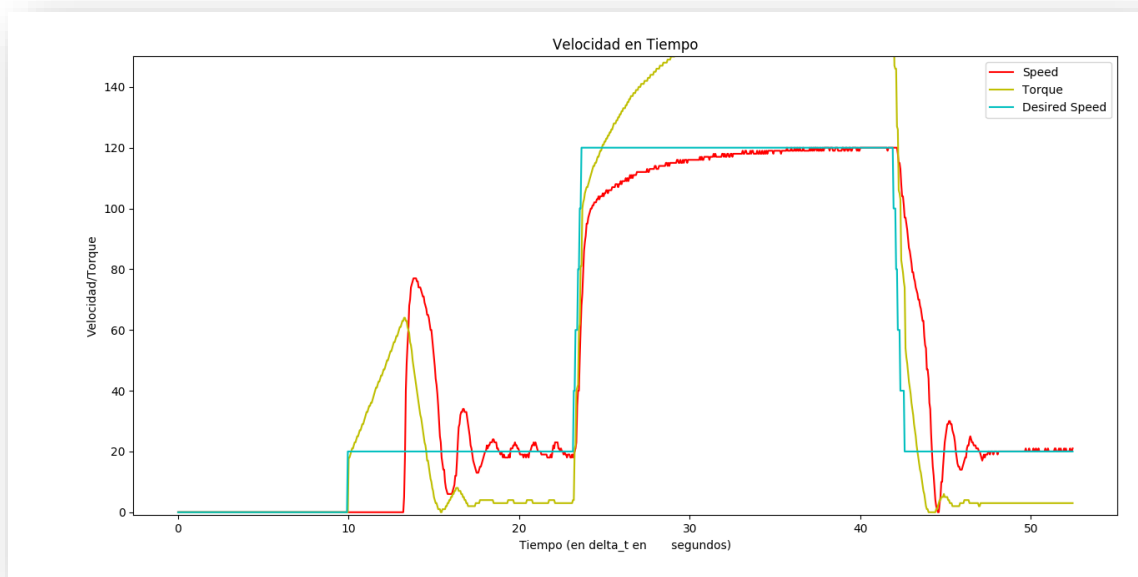
Tanto para el motor UDP como para el simulado se utilizan las variables $k_p=0.873$, $k_i=0.25$, $k_d=0.78$

Con el motor UDP

Consigna: Velocidad inicial 20, con iteraciones de 20 hasta una velocidad final de 120. Repetir hasta el descenso a 20

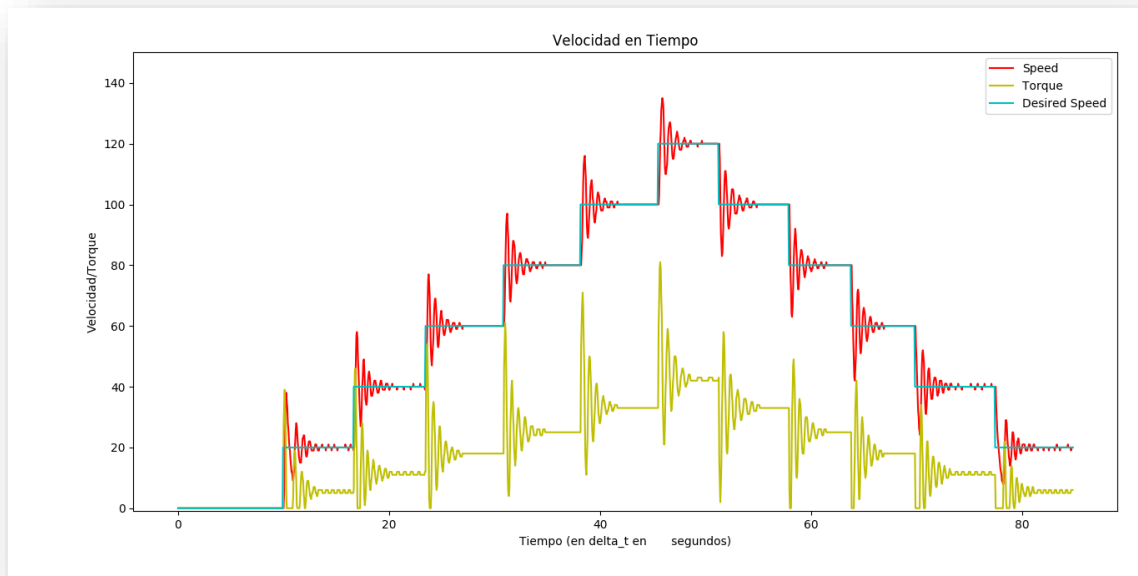


Consigna: Velocidad inicial 0 aumentando hasta 120 para luego descenso hasta 0, sin iteraciones, sino continuo.



Con el motor simulado:

Consigna: Velocidad inicial 20, con iteraciones de 20 hasta una velocidad final de 120. Repetir hasta el descenso a 20



Consigna: Velocidad inicial 0 aumentando hasta 120 para luego descenso hasta 0, sin iteraciones, sino continuo.

