

Firmware

Laboratorio remoto: Viga simplemente apoyada

UNIVERSIDAD DE LA MARINA MERCANTE, CABA, BUENOS AIRES ARGENTINA

leandro.cintioli@alumnos.udemm.edu.ar

luiz.villacorta@alumnos.udemm.edu.ar

pablo.tavolaro@alumnos.udemm.edu.ar

Profesor Tutor: Marcelo Bellotti

Objetivos

- Describir las funcionalidades y componentes que tiene el firmware

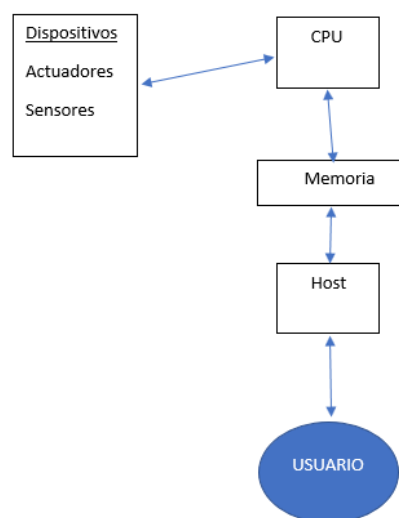
Antecedentes (Ver anexo A)

Documento LRVSA_proyecto final_viga_simplemente_apoyada.

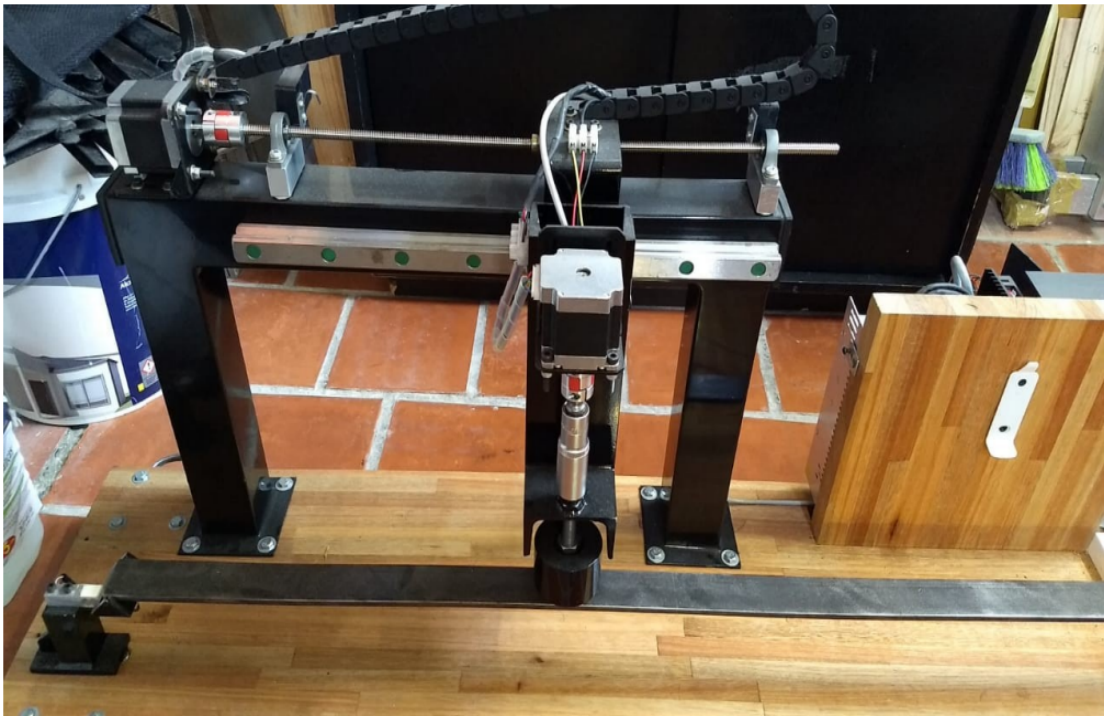
Documento “Proyecto Final 2019 - Banco de prueba - Viga sometida a carga puntual R3”:

Desarrollo

Para poder cumplir con la buenas prácticas de la programación y así darle solidez,escalabilidad y mantenimiento simple al proyecto, se desarrolló la siguiente arquitectura en C++, compilado con el ide arduino.



Donde el usuario se comunica con la cpu mediante un sistema de host que comparte la memoria con la cpu, la cpu está escuchando los cambios en la memoria, y actúa según estos. Con esto se logra la abstracción de todas las partes del sistema en módulos ,que solo son consumidos desde la cpu.



Para establecer la posición del ensayo, se inicializa el punto de referencia cero, con el límite de carrera del motor 1 y luego se mueve la cantidad de pasos necesarios para llegar a la distancia pedida.

Funcionalidades del ensayo Al recibir del host el {cmd:'start'} el ensayo comienza , el logea los estado del mismo por el puerto serie.

```

{
  "cmd": "start",
  "result": "ack"
}
21391 Inicializando motor1. Esperando final de carrera M1
26241 Inicializando motor2. Esperando final de carrera M2
28254 Moviendo el motor 1 cantidad de milimitros
34911 Moviendo el motor 2 hasta leer la fuerza configurada
303.2
Force:Ok
37625 Lectura de fuerza de reaccion 1 en 3 segundos ponga el peso
514.6
42387 Lectura de fuerza de reaccion 1 en 3 segundos ponga el peso
514.5
47144 Lectura del tof
47
51156 ENSAYO TERMINADO, SUERTE!!!
{

```

Funcionalidades genéricas
El firmware consta con lo siguiente sub – sistemas

Configuración

Administra los valores de configuración en la Eeprom

Logeo

Dependiendo de un parámetro de configuración log_level, administra el logeo por el puerto serie del equipo.

```

// {log_level:'0'} log_level:0=desactivado,
// {log_level:'1'} 1=mensajes.
// {log_level:'2'} 2=info control estandar.
// {log_level:'3'} 3=info control arduino plotter

```

Motor

Da soporte al movimiento de los motores

Celda de carga.

Da soporte a las celdas de cargas.

Descripción del firmware

A continuación se describen los módulos básicos que tiene el firmware para controlar el ensayo.

Con un plan a futuro de conectarse con un servidor flask, se plantea el uso de json para la comunicación, se requiere mantener el firmware monolítico dentro de lo posible.

Para evitar lo complejo de la máquinas de estados a-sinconicras, el experimento no puede ser interrumpido mientras se está realizando

El formato del json es:

```
// Lee por el puerto serie parametros de configuracion en formato json.
// {info:'all-params'} Envia todos los parametros en formato json.
// {info:'all-calibration'} Envia todos los parametros en formato json de calibracion de la flexion y el nivel de logeo.
// {info:'version'} Envia la version del firmware.
// {info:'status'} Devuelve el estatus del ensayo.
// {info:'reaction_one'} Devuelve la reaction1 del ensayo.
// {info:'reaction_two'} Devuelve la reaction2 del ensayo.
// {info:'flexion'} Devuelve la flexion del ensayo.
// {info:'st_mode'} Devuelve el modo del ensayo.
// {info:'step_cal'} Devuelve la cantidad de pasos contados para la calibracion de step_k.
// {info:'step_k'} Constante de calibracion de flexion.
// {info:'log_level'} Nivel de logeo por puerto serie.

// {log_level:'0'} log_level:0=desactivado,
// {log_level:'1'} 1=mensajes.
// {log_level:'2'} 2=info control estandar.
// {log_level:'3'} 3=info control arduino plotter.

// {cmd:'start'} Comienza el ensayo.

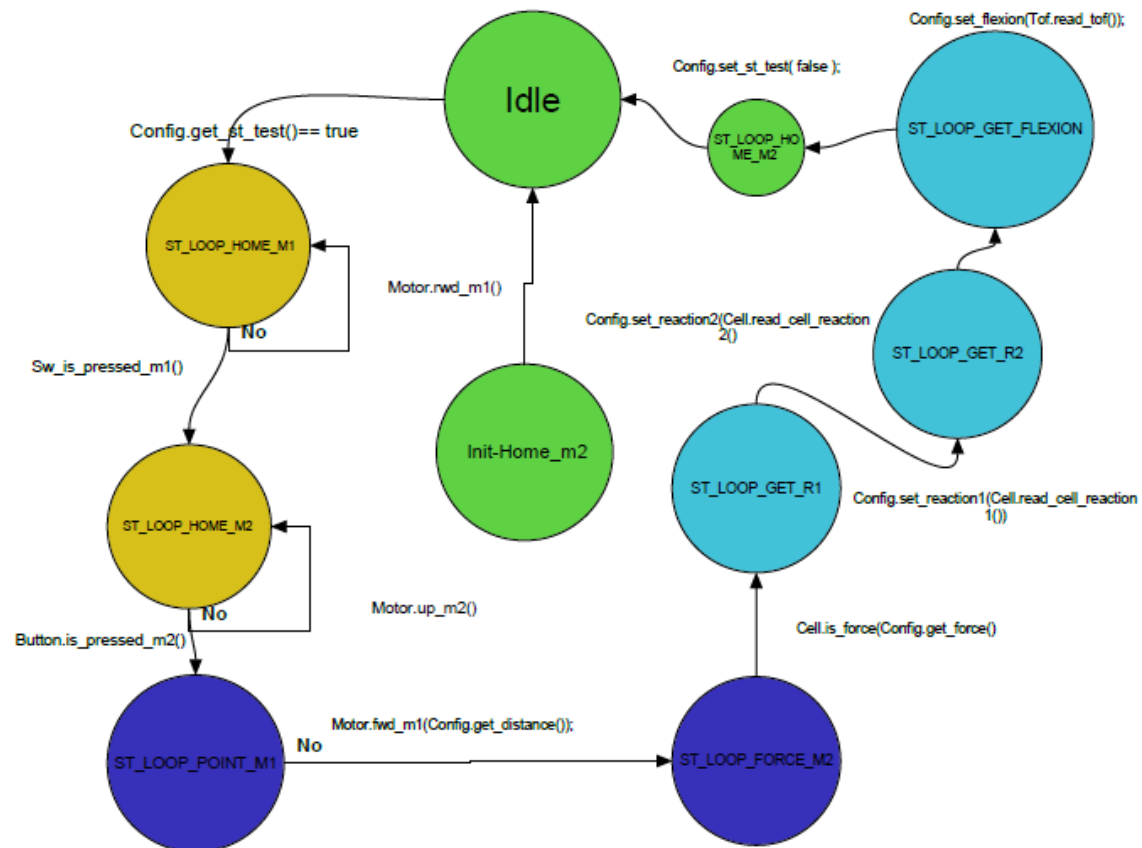
// {m1_fwd:'50'} Mueve 50 mm el motor 1 hacia adelante.
// {m1_rwd:'4'} Mueve 4 mm el motor 1 hacia atras.
// {step_m1_fwd:'200'} Mueve 200 pasos el motor 1 hacia adelante.
// {step_m1_rwd:'200'} Mueve 200 pasos el motor 1 hacia atras.

// {m2_up:'5'} Mueve 5 mm el motor 1 hacia arriba.
// {m2_down:'4'} Mueve 4 mm el motor 1 hacia abajo.
// {step_m2_up:'200'} Mueve 200 pasos el motor 2 hacia arriba.
// {step_m2_down:'200'} Mueve 200 pasos el motor 2 hacia abajo.

// {cdd:'start',data:{distance:'20',force:'306'}}

// {distance:'290'} distance Distancia en mm donde se aplica la fuerza.
// {force:'2000'} force Fuerza a aplicar en g.
// {reaction_one:'1'} reaction_one Fuerza de reaccion uno, en g.
// {reaction_two:'2'} reaction_two Fuerza de reaccion dos, en g.
// {flexion:'0.12630'} flexion Flexion del ensayo, en cm.
// {st_test:'1'} st_test 0 ensayo desactivado.
// st_test 1 ensayo activado.
// {st_mode:'0'} st_mode ST_MODE_TEST 0 ensayo activado.
// ST_MODE_HOME_M2 1 Va al home del motor 2.
// ST_MODE_CELL 2 Lee las celdas de carga.
// {step_cal:'2000'} step_cal Devuelve la cantidad de pasos contados para la calibracion de step_k.
// {step_k:'0.123456'} step_k Constante de calibracion de flexion.
```

Máquina de estados



La variable `st_loop` almacena el estado de la máquina

```
#define ST_LOOP_INIT          0 // Inicializa el programa (carga la configuracion).
#define ST_LOOP_IDLE          1 // Espera la recepción por comando.
#define ST_LOOP_HOME_M1       2 // Busca la referencia del Motor 1.
#define ST_LOOP_HOME_M2       3 // Busca la referencia del Motor 1.
#define ST_LOOP_POINT_M1      4 // Se mueve m1 cantidades de pasos requeridos en mm.

#define ST_LOOP_FORCE_M2       5 // Se mueve m2 hasta que encuentra la fuerza requerida en kilos.
#define ST_LOOP_GET_R1         6 // Lee la celda de carga reaction 1.
#define ST_LOOP_GET_R2         7 // Lee la celda de carga reaction 2.
#define ST_LOOP_GET_FLEXION    8 // Calcula la flexión y la guarda en la configuración
#define ST_LOOP_OFF_TEST       9 // Termino el ensayo.
#define ST_LOOP_MODE_HOME_M2   10 // Se mueve al home 2.
#define ST_LOOP_MODE_CELL      11 // Lee las celdas de carga.
```

El código se puede encontrar en: https://github.com/theinsideshine/flexion_viga

Modos de ejecución

Para poder realizar pruebas durante el desarrollo se cuenta con modos de ejecución

```
{st_mode:'0'}    st_mode    ST_MODE_TEST          0 ensayo activado.
//              ST_MODE_HOME_M2      1 Va al home del motor 2.
//              ST_MODE_CELL          2 Lee las celdas de carga.
```

Comandos del motor

Para poder realizar pruebas durante el desarrollo se escribió soporte para comandos del motor

```
// {m1_fwd:'50'}    Mueve 50 mm el motor 1 hacia adelante.
// {m1_rwd:'4'}     Mueve 4 mm el motor 1 hacia atras.
// {step_m1_fwd:'200'} Mueve 200 pasos el motor 1 hacia adelante.
// {step_m1_rwd:'200'} Mueve 200 pasos el motor 1 hacia atras.


// {m2_up:'5'}      Mueve 5 mm el motor 1 hacia arriba.
// {m2_down:'4'}     Mueve 4 mm el motor 1 hacia abajo.
// {step_m2_up:'200'} Mueve 200 pasos el motor 2 hacia arriba.
// {step_m2_down:'200'} Mueve 200 pasos el motor 2 hacia abajo.
```

Anexo A-referencias

“Proyecto Final 2019 - Banco de prueba - Viga sometida a carga puntual R3”:

 **Proyecto Final 2019 - Banco de prueba - Viga sometida a carga puntual R3.docx**

Proyecto final 2021: laboratorio remoto viga simplemente apoyada

 **LRVSA_proyecto final.docx**


Código fuente

Código en c++ del ensayo: https://github.com/theinsideshine/flexion_viga

Código en python del servidor del ensayo:

https://github.com/theinsideshine/beam_remote_lab_server

Código en Vue del cliente: https://github.com/theinsideshine/flexion_viga

Descripción del firmware:  **Firmware-RLVSA**

Sistema de comunicación:  **Comunicacion-RLVSA**