Informe Final de Proyecto: FLER(Four Leg Explorer Robot)

Integrantes:Alison Lisby, Emmanuel Rodríguez,Kenneth Leiva, Harry Lisby

***Resumen*—Este proyecto tiene como objetivo la aplicación de los conceptos aprendidos de los dispositivos de open source, desde los protocolos de conexión que tienen hasta su capacidades a la hora de controlar otros dispositivos.**

I. Descripción del Proyecto

FLER consiste de un robot explorador capaz de obtener las condiciones ambientales y gases de un espacio reducido, además de transmitirlos a través de una interfaz al usuario con el fin de determinar si un lugar es seguro o no para permanecer o ingresar.

El proyecto de dividió en diferentes partes las cuales se mencionan a continuación.

II. Procedimiento

II-A. Diseño de FLER

- Se diseñó toda la estructura de FLER con ayuda de Fusion Autodesk 360.
- Las partes de aluminio se cortaron con una cortadora CNC y se utilizó Aluminio 6061.
- Además se ensamblaron los piñones para el reductor diseñado en Fusion 360.

II-B. Joystick

Se crea un programa básico que nos permite controlar el desplazamiento y movimiento general de FLER, con el uso de dos Joystick.

Con este programa se quieren controlar 6 ejes distintos:

- Eje X
- Eje Y
- Eje Z
- Eje R (Rotación)
- Eje I (Inclinación)
- Eje G (Giro)

Al crear la librería, se crean cuatro funciones principales:

- readY
- readX
- readYB
- readXB

Las funciones readY y readX se asignan cuando no se presiona el botón del Joystick, mientras que las funciones readXB y readYB se utilizan con el botón presionado.

La declaración de estas funciones y de ciertas variables se observa a continuación.

```
class Joystick {
 public:
   void Joystick();
   uint16_t readY();
   uint16_t readX();
   uint16_t readYB();
   uint16_t readXB();
   uint8_t dir;
   Adafruit_ADS1115 ads;
 private:
   uint16_t _adcY;
   uint16_t _adcX;
   uint16_t _adcYB; \\z
   uint16_t _adcXB; \\g
   int _boton;
   bool _value_boton;
   int _adc1;
   int _adc2;
};
```

A continuación se observa la función readY. Se le asigna un valor de cero a la variable adcY cuando presionamos el botón y cuando no está presionado, se le asigna el valor leído. De esta manera se obtiene el valor del eje Y.

```
uint16_t Joystick::readY(){
   _value_boton = digitalRead(boton);
   if (_value_boton == 1){
     _adcY=ads.readADC_SingleEnded(adc1);
   }
   else{
     _adcY = 13150;
   }
   return _adcY;
}
```

II-C. Condiciones ambientales

Para la parte de la medición de condiciones ambientales tenemos varios componentes:

- Sensor DHT22
- Sensor MQ-2
- LCD Nextion Screen
- 2 NodeMCU micros

Para lo cual cada uno de los micros van a desempeñar dos papeles específicos: Servidor y Cliente.

Servidor: Este micro se va a ubicar en el robot y es el encargado de medir cada uno de los sensores, y mandar los datos de dichos sensores cuando el cliente le hace la petición de los mismos.

Cliente: Este micro se ubica en el control del robot, y este se encarga de hacer las peticiones de datos al servidor y también de enviar dichos datos a la pantalla LCD que también se encuentra en el control.

Para el código del servidor tenemos varias porciones importantes, que se detallan a continuación:

```
void handleRoot() {
String s = "<> Humedad: ";
s += String(hums);
s += " Temperatura: ";
s += String(temps);
s += "Saturacion: ";
s += String(gass);
}
```

En el código anterior vemos como se ejecuta la función handleRoot, la cual me permite a mi enviar la información de cada uno de los sensores en un solo string cuando el cliente lo necesita, toda esta información se manda en formato http. Para el micro del servidor el código a continuación establece el modo de trabajo en el cual para este caso la función WiFi.softAP lo hace.

```
WiFi.softAP(ssid);//, password);
IPAddress myIP = WiFi.softAPIP();
Serial.print("AP IP address: ");
Serial.println(myIP);
server.on("/DataSensores", handleRoot);
server.begin();
```

Como se menciona anteriormente el micro del cliente lleva un código el cual se detalla a continuación:

```
Serial.begin(9600);
delay(10);
WiFi.mode(WIFI_STA);
WiFi.begin("FLER");
while (WiFi.status() != WL_CONNECTED) {
delay(500);
  Serial.print(".");
```

Para el código anterior tenemos que su modo de trabajo va a ser de cliente, la función WiFi.mode (WIFI_STA), difine que este micro va a ser una terminal o dispositivo de la red wireless del micro del Servidor, donde su IP es 192.168.4.1, la cual es la IP default que la define la librería WebServerESP8266.

El cliente dede hacer un request del string llamado “DataSensores”, el cual es construido en el servidor con los datos de cada uno de los sensores, este request se hace por medio de este código detallado a continuación.

```
WiFiClient client;
const int httpPort = 80;
!client.connect("192.168.4.1",
   httpPort);
client.print(String("GET ")
   +"/DataSensores"+" HTTP/1.1\r\n" +
              "Host: " + host + "\r\n" +
              "Connection:
                 close\r\n\r\n");
```

Una vez estabalecida esa comunicación entre el servidor y el cliente, así como el envio del string, tuvimos que hacer la extracción de las partes del string que se requerían y luego convertirlas a float para luego ser convertidas en string de nuevo y ser enviadas a la pantalla LCD ubicada en el control.

```
      while(client.available()){
      String line =
         client.readStringUntil('\r');
      hums =
```

```
        line.substring(14,19).toFloat();
      Serial.println(hums);
```

Como podemos ver la función line.substring() nos permite seleccionar los caracteres que se encuentran en las posiciones de la 14 a la 19, que corresponden al valor en porcentaje de humedad en el ambiente según el sensor DHT22. Después de cada extracción de datos y convertidos a float se tiene que enviar a la pantalla lo cual nos permite según la porgramación de la misma(LCD), escribirlas en un string llamado igual a la variable declarada para los sensores. En esta porción de código corresponde la de Gas.

```
  data = "gass.txt=\"" + String(gass, 1)
     + "\"";
 Serial.print(data);
 Serial.write(0xff);
 Serial.write(0xff);
 Serial.write(0xff);
 data = "gass.txt=\"" + String(gass, 1)
     + "\"";
```

II-D. Control PID

Los movimientos de cada uno de los ejes se realizaron mediante motores DC los cuales se ubicaron en cada una de las articulaciones de las patas, para un total de doce motores. Cada motor es controlado por un microcontrolador STM32F104 por lo cual se requiere de un sistema que sea capaz de suministrar la posición de cada motor y de ahí poder controlar el movimiento. De ahí que se implementó un controlador PID el cual recibe la posición de cada uno de los motores mediante potenciómetros que actúan como "encoders" según el punto de seteo así se ejecuta el movimiento de cada motor junto con el modelo cinemático que se menciona más adelante.

Para el desarrollo del controlador PID se implementó una librería tal y como se detalla a continuación:

```
pidControl::pidControl(double*
    encRead,double* output,double*
    setPoint,double kp,double kd,double
    ki)
:workPID(&_encRead,&_output,&_setPoint,kp,
ki,kd,DIRECT){
  _encRead=*encRead;
  _output=*output;
  _setPoint=*setPoint;
  _kp=kp;
  _kd=kd;
  _ki=ki;
  }
```

Este es el contructor de la clase, donde se detallan las variables a utilizar en el PID. Como se puede observar se están utilizando punteros por lo que vamos a estar trabajando con las posiciones en memoria, ya sea leyendo, escribiendo o solicitando un posición de memoria, según sea necesario. En esta parte se detallan las variables indispensables para el controlador.

```
void pidControl::softwareLimits(double
    minPos=0, double maxPos=4096){
  _minPos = minPos;
  _maxPos = maxPos;
}
```

Está función determina las variables utilizadas para determinar los límites del movimiento de las patas, ya que sin esto podríamos ocasionar sobre giros provocando daños en las diferentes partes del robot.

```
void pidControl::controllerBegin(int
    encoderInput, int pwmOutput, int
    fwdOutput, int revOutput, double
    minOutValue, double maxOutValue,
    double PID_THRESHOLD=15000){
  ENCODER = encoderInput;
  PWM_OUTPUT = pwmOutput;
  FWD_OUTPUT = fwdOutput;
  REV_OUTPUT = revOutput;
  _PID_THRESHOLD = PID_THRESHOLD;

  pinMode(ENCODER, INPUT_ANALOG);
  pinMode(PWM_OUTPUT, PWM);
  pinMode(FWD_OUTPUT,OUTPUT);
  pinMode(REV_OUTPUT,OUTPUT);

  minOut = minOutValue;
  maxOut = maxOutValue;

  pidControl::workPID.SetOutputLimits
      (minOut,maxOut);
  pidControl::workPID.SetMode(AUTOMATIC);
}
```

Acá se detallan las otras variables a utilizar en

el PID, como se observa se detallan las salidas, las entradas, así como los tipos de pines, además se llaman a las otras funciones que forman parte del controlador. Este controlador debe leer los valores del voltaje de cada uno de los potenciómetros, mapearlos y convertirlos a grados para poder retroalimentar este valor al sistema principal y a su vez es el encargado de enviar la señal de control al driver del motor para activar el movimiento, de ahí la importancia de tener todas las variables necesarias debidamente declaradas.

```
void pidControl::run(bool enableAlarm =
    false){
  pidControl::workPID.Compute();

  _encRead = analogRead(ENCODER);

  if((_encRead>_maxPos)){ //Mejorar para
      que retorne al punto maximo
    _setPoint=_maxPos;
  }else if((_encRead<_minPos)){
    _setPoint=_minPos;
  }

  if(_encRead < _setPoint){
    digitalWrite(FWD_OUTPUT,LOW);

    digitalWrite(REV_OUTPUT,HIGH);
  }else if(_encRead > _setPoint){
    digitalWrite(REV_OUTPUT,LOW);

    digitalWrite(FWD_OUTPUT,HIGH);
  }else{
    digitalWrite(REV_OUTPUT,LOW);
    digitalWrite(FWD_OUTPUT,HIGH);
  }

  pwmWrite(PWM_OUTPUT,abs(_output));
}
```

La función “Run.[es] una de las más importantes, ya que es la encargada de todos los cálculos del controlador. Esta función compara el valor de seteo contra la lectura del potenciómetro y según sea el resultado activa o desactiva las señales correspondientes del driver al que se encuentra el conectado el motor. Debemos recordar que la lectura del potenciómetro es convertida a grados por lo que la comparación entre ambas señales se realiza en términos de grados y no de voltaje.

```
double pidControl::getEncoder(){
  return _encRead;
}

double pidControl::getOutput(){
  return _output;
}

double pidControl::getSetpoint(){
  return _setPoint;
}
```

Por último se agregaron estas funciones que son las encargadas de obtener los valores de las variables privadas declaradas para la librería.

II-E. Modelo Cinemático Inverso

El modelo cinemático es el encargado de traducir la geometría física de un dispositivo, de manera que pueda representarse matemáticamente y al mismo tiempo mediante software. La idea principal es que el control sea dado por medio de un número reducido de parámetros que llamaríamos .[ej]esz este modelo matemático automáticamente calcularía los datos necesarios para el control de motores y su posicionamiento.
En el caso que no existiera este sistema se debería programar cada movimiento paso por paso, motor por motor, y cada variación representaría un cambio en el programa, el cuál sería bastante extensivo y tedioso de desarrollar.
Gracias a la combinación de los motores, junto a la retroalimentación dada por potenciómetros se pueden representar diferentes posiciones en forma de ángulos, los cuales permiten realizar cálculos trigonométricos soportados por la plataforma Arduino para su conversión a software.

Eje Z

El eje Z es uno de los ejes principales, o más primitivos, este siempre se verá afectado por los cálculos realizados en los siguientes ejes por lo tanto se verán diferentes modificaciones al mismo a lo largo de la función modeloCinematicoXYZ().

En la siguiente función se puede contemplar los cálculos correspondientes al eje Z, primeramente se calcula el ángulo çita”que representa el ángulo en la parte superior de la pierna, también llamado ”2”. El eje Z representa al único eje que tiene

movimiento vertical, este es el encargado de la variación de la altura del robot.

Para este eje también es necesario calcular la altura en términos del ángulo del eje inferior [3], el cuál se despeja al duplicar el valor de cita. Esto se debe a que la geometría fué diseñada de tal manera que ambas distancias entre articulaciones sean la misma, así el ángulo se comportará de manera proporcional al otro, en este caso dos veces debido a que la posición está dada a 45 grados de la sección superior.

Además de esto es necesario calcular un offset dado por la geometría de la pata, donde hay un movimiento extra, circular, generado por el movimiento del eje superior [1]. A este offset de distancia se le llama Zex debido a que es un extra generado en Z por el movimiento del eje X.

```
//Calculos del eje Z
cita = acos(mediaAltura/P);
cita = cita*RAD_TO_DEG;
citaPrima = 2*cita;

Zex = Xex*tan(beta*DEG_TO_RAD);
convert(Zex,-10,10,10,-10);
```

Eje Y

Este es el encargado de realizar los movimientos horizontales en dirección frontal y trasera del robot, con este será posible controlar movimientos básicos como el caminado y giro.

El código de igual manera se encarga de calcular el ángulo correspondiente para poder realizar dicho movimiento, y este ángulo representará una suma o resta al valor ya encontrado previamente de cita. Por lo tanto este no corresponde a un nuevo eje mecánico si no a uno virtual que está integrado dentro de uno de los ya utilizados, en el código se puede observar que es necesario recalcular el valor previamente obtenido para compensar por el nuevo cambio en el eje.

El ángulo alfa entonces corresponde a el valor necesario para mover la pata en dirección positiva al punto de inicio. Luego se calcula el valor nuevo para el ángulo llamado citaNewY que corresponde al eje superior ”2”de la pata.

Finalmente en base al nuevo ángulo es necesario calcular la nueva altura de Z, esto para compensar por cambiar a ser una componente de la posición original, así como fué mencionado anteriormente.

```
//Calculos del eje Y
alfa = atan(Ypos/Zpos);
alfa = alfa*RAD_TO_DEG;
citaNewY=cita-alfa;

if(citaNewY!=cita)cita=citaNewY;

ZnewY=Zpos/(cos(alfa*DEG_TO_RAD));
```

Eje X

Este corresponde al otro eje de movimiento horizontal perpendicular al Eje Y, su función es mover al robot lateralmente para cumplir algunas funciones especializadas como el balance automático, el giro mejorado y tener control preciso del centro de gravedad del dispositivo.

Al igual que los casos anteriores es necesario calcular un ángulo, en este caso correspondiente al eje del hombro ”1”que no se había modificado anteriormente, de la misma manera que los ejes anteriores para este caso la posición del eje Z debe ser recalculada para mantener la altura establecida al inicio de la función en el eje Z, para ello basta con resolver un triángulo dado por la posición inicial y la final, siendo la final la componente a ser resuelta.

```
//Calculos del eje X
rho = atan(Xpos/ZnewY);
rho = rho*RAD_TO_DEG;

ZnewX =
    (ZnewY+Zex)/cos(rho*DEG_TO_RAD);
    //Con offset Zex
cita = constrain(cita,0,45);
```

Otros ejes

Aún hay más ejes que pueden ser programados para mejorar más aún el funcionamiento y ampliar las capacidades de control de dicho sistema, estos ejes representan el giro, inclinación y rotación. Los cuales aún están pendientes de implementar pero no afectan al funcionamiento necesario del robot,

si no que estos buscan generar un atajo para ciertos movimientos que de lo contrario tendrían que ser programados de manera manual y basados en los ejes X,Y y Z.

Loop principal del modelo cinemático

```
void loop(){
  currentTime = millis();

  serialDecoder();

  if((currentTime-lastTime1)>10){
    modeloCinematicoXYZ(controllerReader(VRy_L),
        controllerReader(VRy_R),
        controllerReader(VRx_R));
    lastTime1=currentTime;
  }

  //send the data
  Front.sendData();
  Rear.sendData();

  //JoysticReads
  if((currentTime-lastTime2)>500){
    Serial.println("xL:
        "+String(controllerReader(VRx_L))+"
        yL:
        "+String(controllerReader(VRy_L))+"
        xR:
        "+String(controllerReader(VRx_R))+"
        yR:
        "+String(controllerReader(VRy_R)));
    lastTime2=currentTime;
  }
}
```

En el programa principal en ejecución se ejecuta principalmente la función del modelo cinemático con una velocidad de cálculo limitada por el mismo programa, pero variable en caso de necesitar menor o mayor tiempo.
Cabe destacar que para esta implementación se prescindió de la utilización de funciones de retraso o pausa como delay(), debido a que los espacios de tiempo y los muestreos necesarios deben de ejecutarse en momentos específicos y ser controlables. Un retraso en la ejecución del programa representaría una alteración en los tiempos establecidos de muestreo y cálculo, generando problemas y datos erróneos en promedios y diferentes interacciones basadas en tiempo.

Se debe destacar también la utilización de la librería EasyTransfer la cuál será discutida en la siguiente sección.

II-F. EasyTransfer

Esta es una librería para comunicación simple basada en estructuras determinadas por el usuario para que no sea necesaria la creación de un protocolo de comunicación entre dos dispositivos cada vez que sea necesaria su interconexión.
Para este caso que se utilizaron hasta cuatro microcontroladores en el robot era necesario tener una manera de comunicarse, y aquí es donde fué de gran utilidad en esta implementación.
Primeramente es necesario generar una estructura con todos los datos a ser transferidos o recibidos, tanto en el emisor como en el receptor esta estructura debe ser la misma, y antes crear una instancia de la librería.

```
EasyTransfer Front;

struct SEND_DATA_STRUCTURE{

  double SSP_IF1, SSP_IF2, SSP_IF3,
      SSP_DF1, SSP_DF2, SSP_DF3;
```

```
};

//dar un nombre a los datos
SEND_DATA_STRUCTURE frontData;
```

Luego de esto se inicializa el puerto serial correspondiente y se definen estos parámentros en la inicialización de la librería.

```
  Serial2.begin(9600);
  Front.begin(details(frontData),
      &Serial2);
```

Finalmente, en el void loop() solo es necesario llamar a la función encargada de enviar los datos, llamada sendData().

```
  Front.sendData();
```

II-G. Control de estabilidad

El control de estabilidad fue pensado para realizarse con un acelerómetro - giroscopio MPU 6050 con el fin de obtener los ángulos de 'pitch' y 'roll', para ello fue necesario implementar dos códigos, uno para la calibración y otro para la medición.

Calibración

Las partes principales del código, dejando de lado el establecimiento de las variables, son las siguientes:

```
void meansensors(){
  long i=0, buff_ax=0, buff_ay=0,
      buff_az=0, buff_gx=0, buff_gy=0,
      buff_gz=0;

  while (i<(buffersize+101)){
    // read raw accel/gyro measurements
        from device
    accelgyro.getMotion6(&ax, &ay, &az,
        &gx, &gy, &gz);

    if (i>100 && i<=(buffersize+100)){
      buff_ax=buff_ax+ax;
      buff_ay=buff_ay+ay;
      buff_az=buff_az+az;
      buff_gx=buff_gx+gx;
      buff_gy=buff_gy+gy;
      buff_gz=buff_gz+gz;
    }
    if (i==(buffersize+100)){
      mean_ax=buff_ax/buffersize;
      mean_ay=buff_ay/buffersize;
      mean_az=buff_az/buffersize;
      mean_gx=buff_gx/buffersize;
      mean_gy=buff_gy/buffersize;
      mean_gz=buff_gz/buffersize;
    }
    i++;
    delay(2);
  }
}
```

El código toma mil cien lecturas del acelerómetro y las promedia, de manera que vamos a obtener el error promedio del sistema que a su vez es lo que necesitamos compensar para poder obtener un valor de cero en la posición inicial.

```
void calibration(){
  ax_offset=-mean_ax/8;
  ay_offset=-mean_ay/8;
  az_offset=(16384-mean_az)/8;

  gx_offset=-mean_gx/4;
  gy_offset=-mean_gy/4;
  gz_offset=-mean_gz/4;

  while (1){
    int ready=0;
    accelgyro.setXAccelOffset(ax_offset);
    accelgyro.setYAccelOffset(ay_offset);
    accelgyro.setZAccelOffset(az_offset);

    accelgyro.setXGyroOffset(gx_offset);
    accelgyro.setYGyroOffset(gy_offset);
    accelgyro.setZGyroOffset(gz_offset);

    meansensors();
```

Luego ese error se divide entre ocho y entre cuatro, que son los valores de sensibilidad del sensor según la especificación del mismo. El resultado obtenido se establece en el MPU 6050 mediante la línea "accelgyro.setAccelOffset()" la cual según la librería que es la encargada de setear el offset para poder obtener la lectura de cero y poder establecer un punto de partida en el sistema.

Obtención de ángulos

Con el MPU 6050 obtuvimos las componentes en X, Y y Z del sistema lo que a través del teorema de pitágoras nos permitió obtener los ángulos correspondientes a la posición.

```
sensor.getAcceleration(&ax, &ay, &az);

 float
    accel_ang_x=atan(ax/sqrt(pow(ay,2) +
    pow(az,2)))*(180.0/3.14);
 float
    accel_ang_y=atan(ay/sqrt(pow(ax,2) +
    pow(az,2)))*(180.0/3.14);
```

Cabe destacar que la implementación total del control de estabilidad no fue posible completarla por temas de tiempo y diversos problemas presentados a lo largo del proyecto, por lo que queda como punto de mejora para el proyecto.

III. Aspectos a Mejorar

Definitivmente en un proyecto tan grande los aspectos a mejorar fueron varios, se detallan a continuación.

- Material utilizado para los piñones
- Control inalámbrico (envío de señales de los joystick)
- Implementación del sensor ultrasónico
- Implementación completa del sistema de estabilización
- Optimización del Código para agilizar el envio de datos entre microcontroladores.
- Implementación de los tres ejes adicionales.
- Drivers con mayor capacidad de control sobre los motores.

IV. Resultados

Los puntos siguientes si se lograron:

- Investigar el uso de acelerómetros para el control de estabilidad. (Se realizó pero no se implementó).
- Desarrollo de todo el modelo cinemático.
- Comunicación entre los microcontroladores secundarios y el primario para la movilización de cada una de las patas.
- Instalar el sensor DTH22 en una placa arduino UNO y desarrollar el código correspondiente para el monitoreo de temperatura y humedad relativa.
- Instalar el sensor MQ-2 en la misma placa y desarrollar el código respectivo para el monitoreo de gases peligrosos en el ambiente.
- Investigar y realizar pruebas con el microcontrolador ESP8266 NodeMCU para la transmisión de datos por internet.
- Desarrollar el código correspondiente para el control de los motores y unidad principal mediante un modelo cinemático y control PID, utilizando el microcontrolador STM32F103.

De acuerdo a lo establecido desde el inicio del proyecto los siguientes puntos no se lograron:

- Instalar el sensor ultrasónico HC-SR04 en una placa arduino UNO y desarrollar el código respectivo para detectar obstáculos.
- Instalar un sensor infrarrojo con su respectivo código para detección de obstáculos.
- Determinar cual es el mejor sensor en términos de sensibilidad y respuesta ante la presencia de objetos.
- Analizar los datos y validar la exactitud de los mismos comparándolos contra un tercer elemento (medidor de condiciones ambientales).
- Investigar la mejor opción para el mapeo de los datos a través de internet por ejemplo “pubnub” ó “Cayenne”.
- Eventualmente, instalar una cámara y configurarla através de una plataforma raspberry pi 3.
- Investigar y configurar la opción de video de la cámara de manera que se pueda acceder por internet.