

ELO 322

Redes de Computadores I

****Análisis y aplicación de redes de computadores en robótica de exploración

**Alumnos:**

**Gabriel Fuentes Vergara 201604009-9**

**Matias Fuentes Pfeil 201604157-5**

**RESUMEN:**

En este documento se presenta el desarrollo de una plataforma robótica con control remoto usando el protocolo TCP/IP. Se describe las características físicas, mecánicas y eléctricas de la plataforma sobre la que se desarrolla ROVER Explorador y el modelo usado para su control con sus características. Se explica la decisión de la arquitectura de software empleada y como se implementó.

Además, se incluirán referencias y anexos tanto de las características y códigos propios de la plataforma, como de la “inspiración” del proyecto, en este caso, por el origen de los modelos “*Rover*” o también llamados Astro-móviles utilizados por las organizaciones aéreo-espaciales al rededor del mundo para explorar terrenos distantes, como otros planetas.

Finalmente, se hará un énfasis en la aplicación de contenidos del ramo Redes de Computadores I en la estructura de las comunicaciones involucradas dentro de la plataforma.

**1 INTRODUCCIÓN**

La tecnología nos ha permitido alcanzar diversas metas que antiguamente solíamos describir como “imposibles”. En 1160 se creó el primer dispositivo orientador llamado Brújula, en 1765 se inventó la primera máquina a vapor capaz de transformar la energía a potencia mecánica[[1]](#footnote-1), la creación de la primera red de computadores alrededor de 1965[[2]](#footnote-2), y uno de los más destacados en la historia de la humanidad es el primer paso del hombre en la Luna en 1969 con el Apolo 11[[3]](#footnote-3). Actualmente, el ser humano está en busca de conocimiento en el espacio y, debido a diversos factores que influyen en nuestra fisiología, no podemos extraer dicha información presencialmente. Es por esto que debemos utilizar diversas tecnologías para extraer, de forma remota, datos de interés y es acá donde las redes de computadores toman el papel protagonista.

 **(Pequeña introducción a ROVER)**

Las redes de computadores permiten establecer un proceso de transmisión de datos de un dispositivo a otro. Éste se compone de cinco capas: Aplicación, Transporte, Red, Enlace de Datos y Capa Física. El objetivo de las capas es establecer una estructura que proporciona estabilidad al sistema, ya que si se modifica una de las capas nombradas el resto quedará intacto y sin necesidad de algún cambio. La Capa en la que se hace énfasis en este proyecto es en la Capa de Aplicación y Transporte. Sin embargo, todas las capas son parte fundamental para que el ROVER Explorador funcione correctamente

El objetivo de ROVER Explorador es el diseño, construcción, implementación y depuración de una plataforma controlada remotamente y de streaming de video. Esto se consigue trabajando tanto en el área de la electrónica, como de la informática.

En este informe se describen cada uno de los procesos que permiten el funcionamiento del ROVER Explorador y cada una de sus estructuras. Primero, se habla de la plataforma y el modelo físico del robot. Después, se hará un énfasis en el interés del ramo con los códigos pertinentes y finalmente se hablara del aporte de éste en la actualidad.

**2 PLATAFORMA**



***Figura 1.*** *Plataforma Dagu Wild Thumber 6WD.*

Todo el desarrollo descrito en éste documento fue hecho sobre ésta plataforma robótica. A pesar de eso, gran parte de la implementación de software es independiente de la plataforma.

Los criterios usados, tales como: abrupto, terreno irregular y otros, son respecto a las dimensiones de la plataforma.

**2.1 CARACTERÍSTICAS FÍSICAS**

La plataforma Dagu Wild Thumber 6WD cuenta con seis ruedas, cada una fija al eje de un motor DC (corriente continua).



***Figura 2***. *Diseño de par de motores Wild Thumber 6V DC.*

Entre las características a importantes, destacan el sistema de suspensión de cada par de motores, permitiendo así el paso por terrenos abruptos.

**2.2 ELECTRÓNICA**

La plataforma usa motores Wild Thumber 6V DC . Éstos motores trabajan idealmente a 6[V], con rango de tolerancia de 2[V] a 7.5[V] como mínimo y máximo respectivamente, y un máximo de corriente de 5.5[A]. Soportan un torque de bloqueo de 8.8[Kg/cm], que es cuando consumen el máximo de corriente. Llegan a las 10.000[RPM].



***Figura 3.*** *Conexión de modo dual de HB-25.*

Los motores son alimentados por dos puentes H HB-25[7], cada uno controlando 3 motores.

Los HB-25 en conexión dual se controlan enviando una señal PWM al primero de la serie.

 La señal de control es generada por una placa de desarrollo Arduino Mega. Se le carga un programa escrito en lenguaje C que implementa un protocolo de comunicación por medio de librerías de *Arduino* para el control de dos motores.

 El Arduino es alimentado por una fit-PC3 a bordo, un computador portátil. La fit-PC3 tiene,entre otras cualidades, un disco duro SSD. Esto último es importante, ya que resiste movimientos bruscos, tales como golpes o caídas abruptas de la plataforma. Además, se le pude conectar un antena wifi. Es la fit-PC3 la que actúa como servidor, recibiendo instrucciones desde un cliente que envíe mensajes con la estructura de un socket TPC/IP, programado en Python.

**2.3 MODELO TEÓRICO DE CONTROL**

El modelo está basado en que para el control se tienen 2 motores virtuales, un motor izquierdo y un motor derecho, ambos formados por los tres motores del lado correspondiente.



***Figura 4.*** *Representación de potencias en cada motor virtual*

La figura 4 muestra una representación gráfica de ambos motores. Las potencias se distribuyen según un tiempo de *delay(),* el cual define el período de la señal PWM para cada puente H.

Éste modelo permite un control sencillo desde un joystick de juego común, ya que con sólo indicar un par (x,y) se puede generar ambos tiempos de señal.

**3 SOFTWARE DE CONTROL**



***Figura 5.*** *Diagrama de bloques de ROVER Explorador.*

La figura 5 representa a alto nivel (alto grado de abstracción) la conexión realizada para el control de la plataforma. El cliente envía los comandos que luego se ejecutarán en la plataforma.

Ambas partes de la comunicación fueron desarrolladas usando el lenguaje Python, sobre el sistema GNU/Linux, aprovechando tanto bibliotecas estándar del lenguaje, bibliotecas propias de GNU/Linux y comunicación de procesos, pipelines, estándar de los sistemas UNIX.

**3.1 SEPARACIÓN EN PROCESOS**

Respecto a los procesos involucrados, podemos destacar 3:

1.- Lectura de valores de Joystick por medio de la librería Pygame, de Python y post-envío, de valores convertidos por medio de una escala conveniente, a través de un Socket TCP/IP a la fit-PC3, sirviendo de canal una red wifi por un router configurado previamente.

2.- Recepción de valores desde la fit-PC3 (Actual servidor en la conexión Socket), y envío por medio de comunicación Serial USB al microcontrolador Arduino Mega.

3.- Luego, se envían los valores al puente H “HB-25” para la distribución de las correspondientes potencias a cada motor virtual.

**4 CONCLUSIÓN**

Finalmente, quisimos mencionar un punto que pensamos, es esencial destacar y es la importancia de la aplicación de contenido aprendido en el curso en este proyecto. Analizando las componentes de la estructura central del robot, vemos que aplicando la configuración de *Sockets* a la comunicación entre computadoras, pudimos establecer una línea de comunicación, que para nuestra necesidad, es “infalible” y su eficiencia sólo quedará determinada por la calidad del medio de difusión utilizado. Esto se explica de la siguiente manera: los códigos escritos para la comunicación son estándar, y no es necesario alterarlos más. Desde el lenguaje de redes de computadores, podemos decir que la capa de aplicación y la capa de transporte están “fijas” (listas para usar, sin ningún cambio necesario) en el transporte de paquetes con datos de ubicación y movimiento del robot. Entonces, todo (distancia de funcionamiento, rapidez de comunicación, etc) “dependerá” del medio de difusión o capa de enlace (calidad y distancia que abarque el router usado).

**Anexos**

En el siguiente anexo, se incluyen 3 códigos/ejemplos para el modelo planteado en la plataforma *Rover*.

En orden, de arriba a abajo, son:

1.- Código Python del cliente

2.- Código Python del Servidor

3.- Código en C++/Arduino para el Microcontrolador

1)

import pygame,sys,time,socket

from pygame.locals import\*

def transformar\_senal(axis):

 if axis == -1:

 senal\_final = '1000\n'

 elif -1 < axis <= -0.8:

 senal\_final = '1100\n'

 elif -0.8 < axis <= -0.6:

 senal\_final = '1200\n'

 elif -0.6 < axis <= -0.4:

 senal\_final = '1300\n'

 elif -0.4 < axis <= -0.2:

 senal\_final = '1400\n'

 elif -0.2 < axis < 0:

 senal\_final = '1450\n'

 elif axis == 0:

 senal\_final = '1500\n'

 elif 0 < axis <= 0.2:

 senal\_final = '1600\n'

 elif 0.2 < axis <= 0.4:

 senal\_final = '1700\n'

 elif 0.4 < axis <= 0.6:

 senal\_final = '1800\n'

 elif 0.6 < axis <= 0.8:

 senal\_final = '1900\n'

 elif 0.8 < axis < 1:

 senal\_final = '1950\n'

 elif axis == 1:

 senal\_final = '2000\n'

 return senal\_final

pygame.init()

pygame.joystick.init()

joystick = pygame.joystick.Joystick(0)

joystick.init()

interval = 0.05

loopQuit = False

skt = socket.socket()

skt.connect(("192.168.2.2",6666))

while loopQuit == False:

 axis\_izq\_ver = round(joystick.get\_axis(1),3)

 axis\_der\_ver = round(joystick.get\_axis(3),3)

 print axis\_izq\_ver

 print axis\_der\_ver

 izq = transformar\_senal(axis\_izq\_ver)

 der = transformar\_senal(axis\_der\_ver)

 msje = transformar\_senal(axis\_izq\_ver).strip() + transformar\_senal(axis\_der\_ver).strip() + '\n'

 print msje

 skt.send(msje)

 for event in pygame.event.get():

 if event.type == QUIT:

 loopQuit = True

 elif event.type == pygame.KEYDOWN:

 if event.key == pygame.K\_ESCAPE:

 loopQuit = True

 time.sleep(interval)

skt.close()

pygame.quit()

sys.exit()

2)

import sys,time,serial,socket

 while True:

 try:

 skt = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) #Se instancia el objeto socket

 skt.bind(("",6666))

 skt.listen(1)

 sclient, addr = skt.accept()

 arduino = serial.Serial('/dev/ttyUSB0',9600)

 interval = 0.01

 while True:

 entrada = sclient.recv(1024) #Se recibe msje y cant de bytes a recibir

 print entrada

 arduino.write(entrada)

 time.sleep(interval)

 sclient.close()

 skt.close()

 arduino.close()

 sys.exit()

 except:

 print "Error"

3)

int n1,n2;

String izq,der;

String inputString = "";

boolean stringComplete = false;

void setup() {

 pinMode(6,OUTPUT);

 // initialize serial:

 Serial.begin(9600);

 // reserve 200 bytes for the inputString:

 inputString.reserve(200);

}

void loop() {

 // print the string when a newline arrives:

 if (stringComplete) {

 Serial.println(inputString);

 izq = inputString.substring(0,4);

 der = inputString.substring(4);

 n1 = izq.toInt();

 n2 = der.toInt();

 digitalWrite(6,HIGH);

 delayMicroseconds(n1);

 digitalWrite(6,LOW);

 delayMicroseconds(1100);

 digitalWrite(6,HIGH);

 delayMicroseconds(n2);

 digitalWrite(6,LOW);

 delayMicroseconds(5250);

 // clear the string:

 inputString = "";

 stringComplete = false;

 }

}

void serialEvent() {

 while (Serial.available()) {

 // get the new byte:

 char inChar = (char)Serial.read();

 // add it to the inputString:

 inputString += inChar;

 // if the incoming character is a newline, set a flag

 // so the main loop can do something about it:

 if (inChar == '\n') {

 stringComplete = true;

 }

 }

}

1. https://sites.google.com/site/latecnologiaysuhistoria/hitos-tecnologicos-historicos [↑](#footnote-ref-1)
2. https://www.zakon.org/robert/internet/timeline/ [↑](#footnote-ref-2)
3. https://es.wikipedia.org/wiki/Apolo\_11 [↑](#footnote-ref-3)