



UNIVERSIDAD TECNOLÓGICA DE PANAMÁ

DISEÑO Y CONTROL DE ROBOTS

DR. RONY CABALLERO

MODELADO Y CONTROL DEL ROBOT MÓVIL ROBOTNIK SUMMIT XL

INTEGRANTES:

LUCAS ROSAS

MIGUEL FUENTES

CARLOS SAMANIEGO

RANGEL ALVARADO

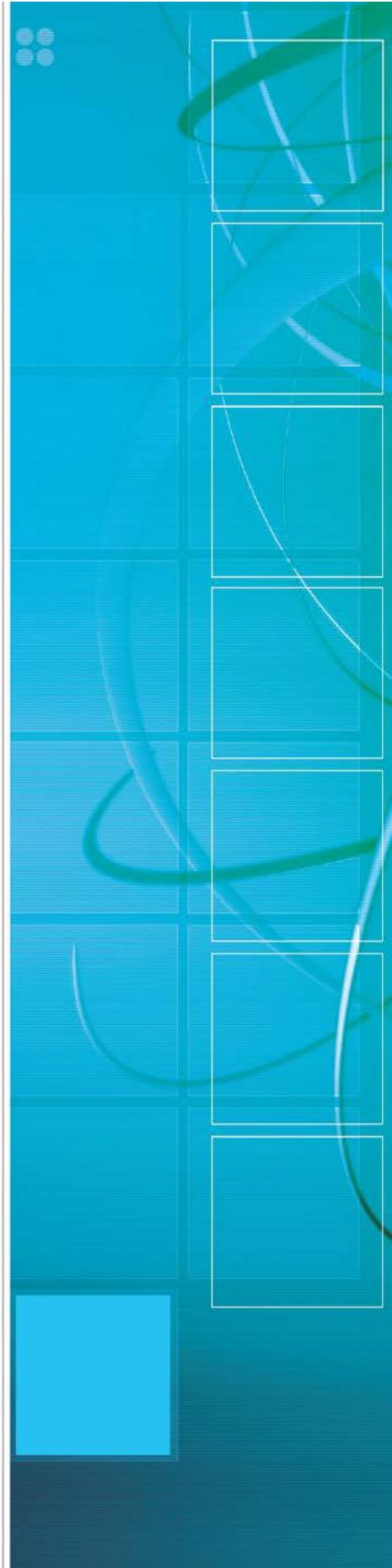
JOAQUÍN VALENCIA

El siguiente documento redacta la manera de realizar aplicaciones con un robot diferencial “skid-steer”, el rover de la compañía Robotnik denominado Summit XL.

Primeramente se estudian los diferentes modelos cinemáticos del robot para encontrar su relación de movimiento. Estudiado el modelo cinemático, se construye un controlador de trayectoria realizado en Matlab.

Seguido se realiza el estudio del modelo dinámico del robot y se establecen las ecuaciones de la dinámica del movimiento del robot a través de las matrices de masa, coriolis y fuerzas gravitatorias.

Finalmente, se realiza la integración de la PC instalando ROS (Robot Operating System), versión Fuerte, se ajusta el sistema en red y se prueban los controladores de velocidad del robot además de la cámara como opción de navegación.



1	Contenido	
2	Descripción del Summit XL.....	3
2.1	Especificaciones técnicas	4
3	Modelo cinemático	4
3.1	Simulaciones	8
3.2	Código del modelo cinemático.....	12
4	Modelo dinámico	13
5	Controlador de seguimiento de trayectoria de un robot diferencial.....	14
5.1	Archivos del controlador.....	15
5.2	Explicación del controlador.....	15
5.3	Inputs del usuario.....	15
5.4	Pre inicialización.....	17
5.5	Inicialización de variables.....	18
5.6	Bucle del controlador	19
5.7	Simulaciones	21
5.7.1	Seguimiento a Curva1.mat	22
5.7.2	Seguimiento a Curva2.mat	23
5.7.3	Seguimiento a Espiral.mat.....	24
5.7.4	Seguimiento a Ocho.mat	25
5.7.5	Seguimiento a Circulo.mat	26
5.8	Efecto de la limitación de velocidad	27
5.9	Efecto de la limitación de los ángulos al rango $-\pi \leq \theta \leq \pi$	28
5.10	El peor de los casos (sin ningún tipo de limitación)	29
5.11	Restricción de velocidad a valores positivos.....	30
5.12	Códigos completos	32
6	Prueba remota desde una terminal telnet con el robot summit XL	41
7	Instalación de ros fuerte en Ubuntu 11.10 (ONEIRIC OCELOT).....	43
8	Controlador de Velocidad del Summit XL (Summit_xl_controller.h).....	62
9	Controlador de Velocidad del Summit XL (summit_xl_controller.cpp)	65
10	Descripción del controlador del summit XL.....	70
11	Conclusiones.....	75
12	Referencias	76

2 Descripción del Summit XL

La plataforma móvil Summit XL, posee cinemática diferencial (skid-steering) basada en 4 motores de alto rendimiento. Cada rueda integra un motor sin escobillas y encoder. La odometría se calcula haciendo uso de los cuatro encoders, y un sensor angular de alta precisión montado en el interior del chasis.

Gracias a su fuerte estructura mecánica, esta plataforma móvil puede transportar cargas mucho más pesadas que el robot Summit estándar (la versión que precede al Summit XL). Posee, además, amortiguadores con potentes posibilidades que pueden ser montados en varias posiciones para modificar el espacio libre entre el chasis y el suelo (clearance) del robot.

El robot puede navegar autónomamente o ser teleoperado mediante una cámara Pan-Tilt-Zoom, que transmite vídeo en tiempo real.

Los accesorios estándar incluyen el escáner láser Hokuyo y kits RTK-DGPS. Así mismo, posee conectividad interna (USB; RS232, GPIO y RJ45) y externa (USB 12 y 24 VDC) para poder acoplar fácilmente todo tipo de componentes.

El framework ROS define una arquitectura bien estructurada e incluye cientos de paquetes de usuario y conjuntos de paquetes llamados pilas (stacks), que implementan un gran número de componentes y de algoritmos como localización, cartografía GIS, planificación, manipulación, percepción, etc.

Esta característica simplifica el ciclo de desarrollo del software y permite una fácil integración, así como la reutilización de componentes de software sean drivers de dispositivos o los más avanzados algoritmos de visión, SLAM, planificación, swarming, etc.

El Summit XL es ideal para aplicaciones de investigación, vigilancia, militares, monitorización remota, acceso remoto a entornos.



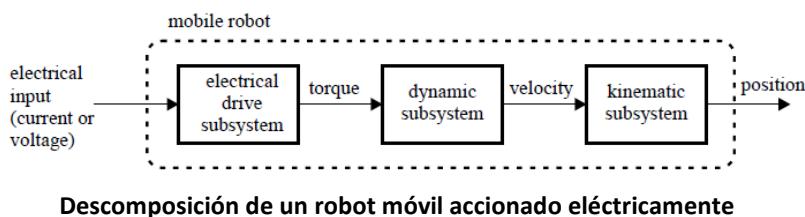
Summit XL

2.1 Especificaciones técnicas

Mecánicas	
Dimensiones	693 x 626 x 417 mm
Peso	30 Kg
Capacidad de carga	20 Kg
Velocidad	3 m/s
Clase de protección	IP53/Opciones de IP66
Sistemas de Tracción	4 ruedas
Autonomía	180 minutos
Baterías	8x3.3V LiFePO4
Motorización	4x250 W servomotores brushless
Rango de temperatura	0° a +50° C
Máxima pendiente	45°

Control	
Controlador	Arquitectura abierta ROS PC empotrado con Linux Real Time
Comunicación	WiFi 802.11n
Conectividad	Interior: USB, RS232, GPIO y RJ45
Velocidad	Exterior: USB y toma de 12 VDC

3 Modelo cinemático



El Summit XL es un skid-steering mobile robot (SSMR) o robot móvil de dirección deslizante. Los SSMR son considerados robot móviles de todo terreno, debido a su robusta estructura mecánica. Ellos pueden trabajar en los peores entornos ambientales, y pueden ser usados para explorar el espacio.

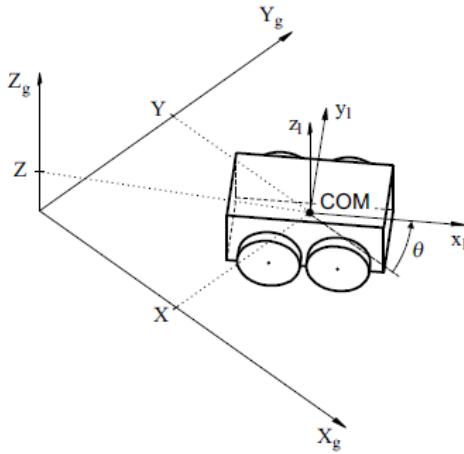
El “steering” de un SSMR se logra manejando diferencialmente los pares de ruedas a cada lado del vehículo.

Aunque el esquema de “steering” proporciona algunas ventajas mecánicas, el control de un SSMR es una tarea difícil, debido a que las ruedas deben derrapar lateralmente al seguir una trayectoria curva.

Si la proyección de centro instantáneo de rotación (ICR) a lo largo del eje X es muy grande, el vehículo puede perder estabilidad, debido al deslizamiento.

Debido al derrape lateral, la restricciones de velocidad que ocurren en un SSMR son muy diferentes de otros plataformas móviles en las cuales no se supone deslizamiento. Esto implica que el control de este robot a nivel cinemático no es suficiente, y en general demanda el uso de algoritmos de control a nivel dinámico también.

Para considerar el modelo cinemático del Summit_XL, en primer lugar se asume que el robot está localizado en una superficie plana, con base ortonormales inerciales (X_g, Y_g, Z_g) , como se muestra en la fig.



Summit_XL en un marco de referencia inercial

Asignamos un marco de coordenadas locales en el centro de masa del robot denotado por (X_l, Y_l, Z_l) . De acuerdo a la fig. 1 las coordenadas del centro de masa en el marco inercial pueden escribirse como $\text{COM} = (X, Y, Z)$. Considerando solo el movimiento en un plano, la coordenada Z del centro de masa (COM) es constante.

Ahora suponemos que el robot se mueve en un plano con una velocidad lineal expresada en coordenadas locales como $v = [v_x \ v_y \ 0]^T$, y rota con un vector de velocidad angular $\omega = [0 \ 0 \ \omega]^T$. Si $q = [X \ Y \ \theta]^T$ es el vector de estado que describe las coordenadas generalizadas del robot, luego $\dot{q} = [\dot{X} \ \dot{Y} \ \dot{\theta}]^T$ denota el vector de velocidades.

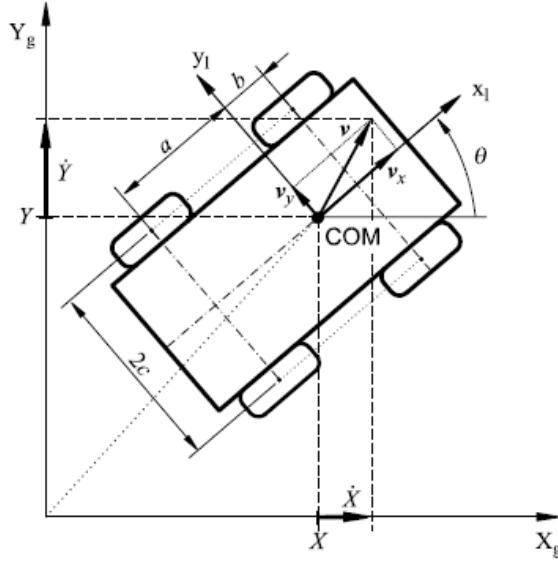
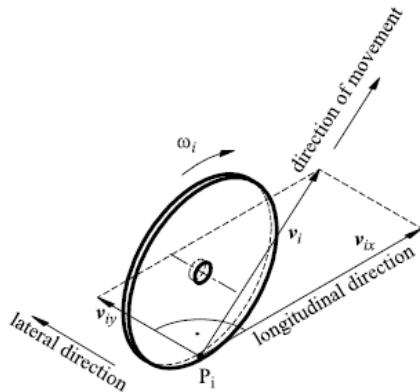


Diagrama de cuerpo libre

De la fig.2 se puede observar que las variables \dot{X} y \dot{Y} están relacionadas a las coordenadas locales de velocidad como sigue:

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix}$$

Como el movimiento es planar también podemos escribir $\dot{\theta} = \omega$. La ecuación (1) no supone ninguna restricción en el movimiento en un plano del robot, ya que describe solamente la cinemática de cuerpo libre del robot. De este modo es necesario también analizar la relación entre las velocidades de las llantas y las velocidades locales. Supóngase que la i -th llanta rota con una velocidad angular $\omega_i(t)$, donde $i = 1, 2, \dots, 4$, las cuales pueden ser una entrada de control. Por simplicidad del análisis el ancho de las llantas es despreciado, y se asume que está en contacto con el plano en un punto P_i como se muestra en la fig.

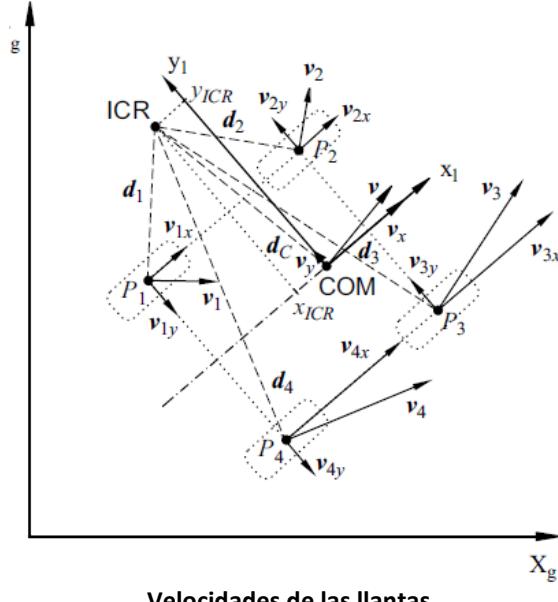


Velocidad de una llanta

Considerando solo el movimiento en un plano, y asumiendo que no existe desplazamiento lateral, podemos escribir:

$$v_x = r_i w_i$$

Ahora bien, en la figura siguiente se da una representación de las velocidades de las cuatro llantas del vehículo.



Las distancias d_1, d_2, d_3, d_4 , se miden del punto de contacto de las llantas al centro instantáneo de rotación. Por geometría podemos determinar que:

$$\begin{aligned}d_{1x} &= d_{2y} \\d_{3y} &= d_{4y} \\d_{1x} &= d_{4x} \\d_{2x} &= d_{3x}\end{aligned}$$

Por lo tanto, las siguientes relaciones entre las velocidades pueden ser escritas:

$$\begin{aligned}V_L &= v_{1x} = v_{2x} \\V_R &= v_{3x} = v_{4x} \\V_F &= v_{2y} = v_{3y} \\V_B &= v_{1y} = v_{4y}\end{aligned}$$

Finalmente, podemos escribir la siguiente relación:

$$n = \begin{bmatrix} v_x \\ w \end{bmatrix} = \begin{bmatrix} \frac{w_L + w_R}{2} \\ \frac{-w_L + w_R}{2c} \end{bmatrix}$$

Donde n , es una variable de control a nivel cinemático, y la velocidad del robot viene dada por:

$$\dot{q} = S(q)n$$

Donde $S(q)$, viene dada por:

$$S(q) = \begin{bmatrix} \cos \theta & X_{ICR} \sin \theta \\ \sin \theta & -X_{ICR} \cos \theta \\ 0 & 1 \end{bmatrix}$$

De la ecuación anterior, se puede observar que la posición del X_{ICR} influye en el movimiento del robot. Teóricamente, puede moverse durante el movimiento del robot, sin embargo si X_{ICR} sale de la distancia entre los ejes, el robot desliza y pierde estabilidad. Para resolver este problema se introduce una restricción matemática no holonomica, descrita por:

$$v_y = x_0 \dot{\theta}$$

Donde x_0 , denota una coordenada fija del ICR , la cual debe elegiré como:

$$x_0 \in (-a, b)$$

Por último $S(q)$, se puede escribir como:

$$S(q) = \begin{bmatrix} \cos \theta & x_0 \sin \theta \\ \sin \theta & -x_0 \cos \theta \\ 0 & 1 \end{bmatrix}$$

Como x_0 es un valor arbitrario lo elegimos cero para simplificar el modelo y obtener que el modelo cinemático del summit XL, se aproxime al de un robot diferencial de dos ruedas, cuyas ecuaciones se describen a continuación:

$$X(t) = X(0) + \int_0^t r \frac{w_R + w_L}{2} \cos \theta dt$$

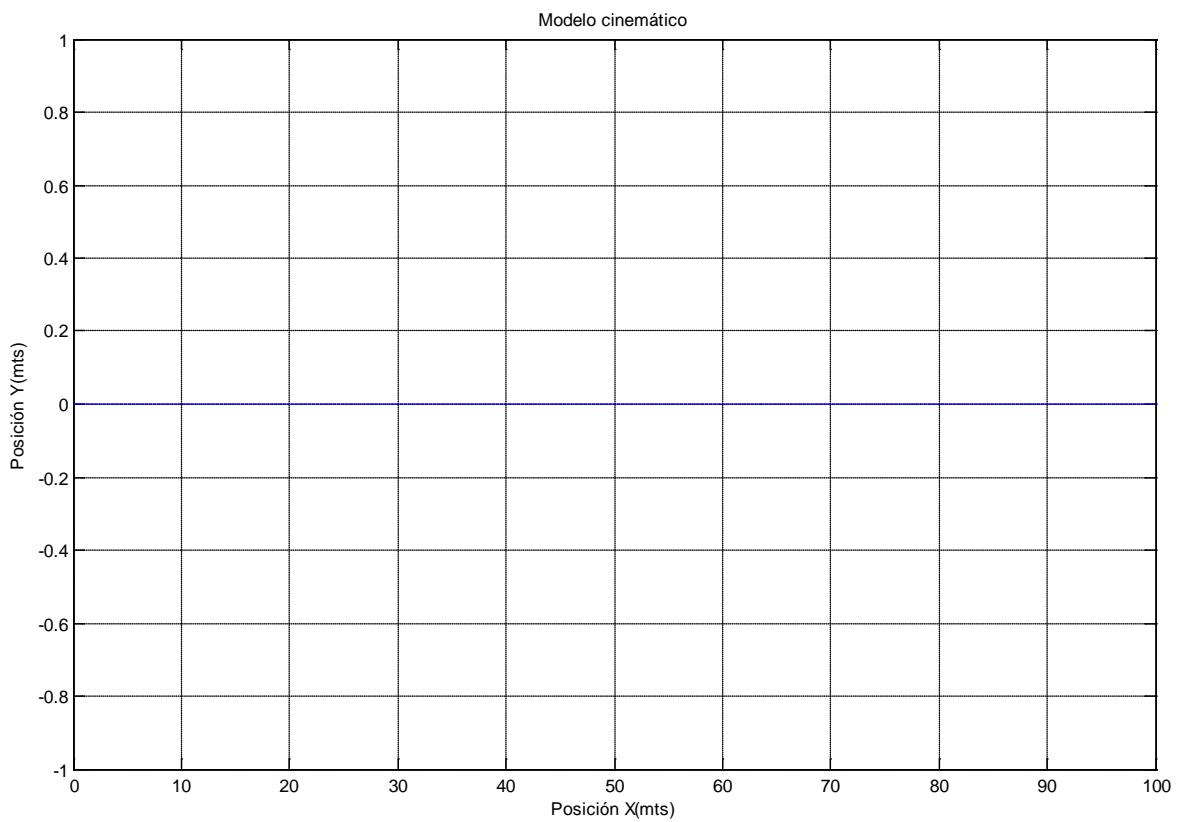
$$Y(t) = Y(0) + \int_0^t r \frac{w_L - w_R}{2} \sin \theta dt$$

$$\theta(t) = \theta(0) + \int_0^t r \frac{w_R - w_L}{2C} dt$$

Las entradas de control del modelo cinemático son las velocidades de las ruedas respectivamente, más adelante se empleara este criterio para desarrollar un controlador de posición.

3.1 Simulaciones

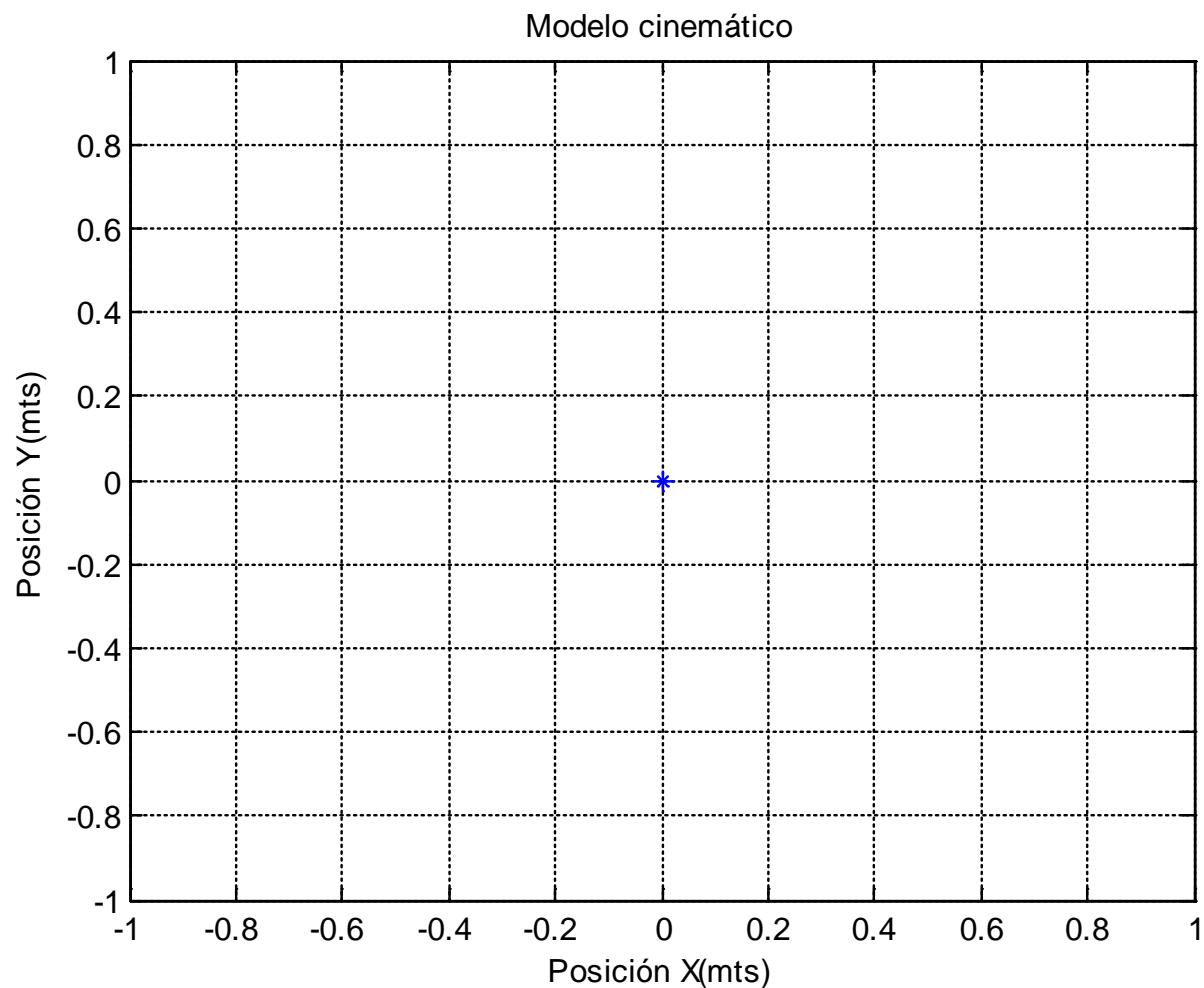
$$WR = 1 \frac{rad}{s}; WL = 1 \frac{rad}{s}$$



Simulación con velocidades de igual magnitud

Para velocidades angulares iguales el robot, mantiene una trayectoria en línea recta.

$$WR = 1 \frac{rad}{s}; WL = -1 \frac{rad}{s}$$

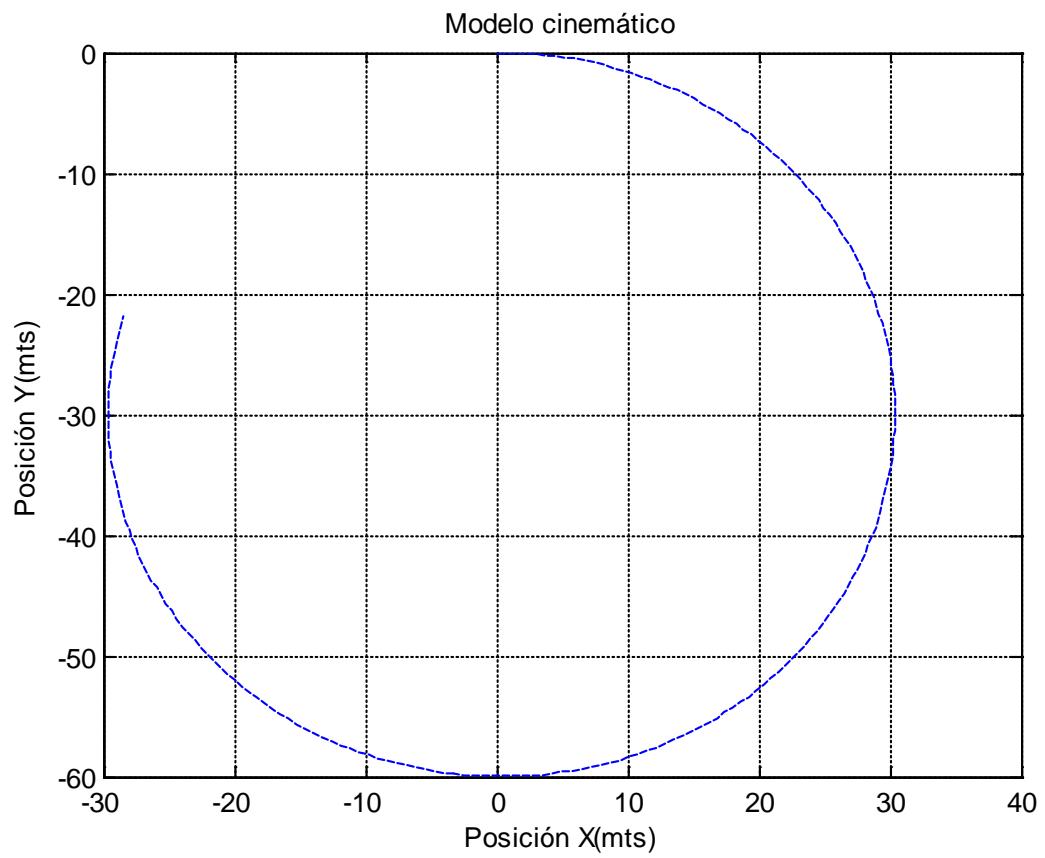


Simulación con velocidades de igual magnitud, y dirección opuesta

Velocidades de igual magnitudes, pero en sentido opuesto no modifican la posición inicial del robot.

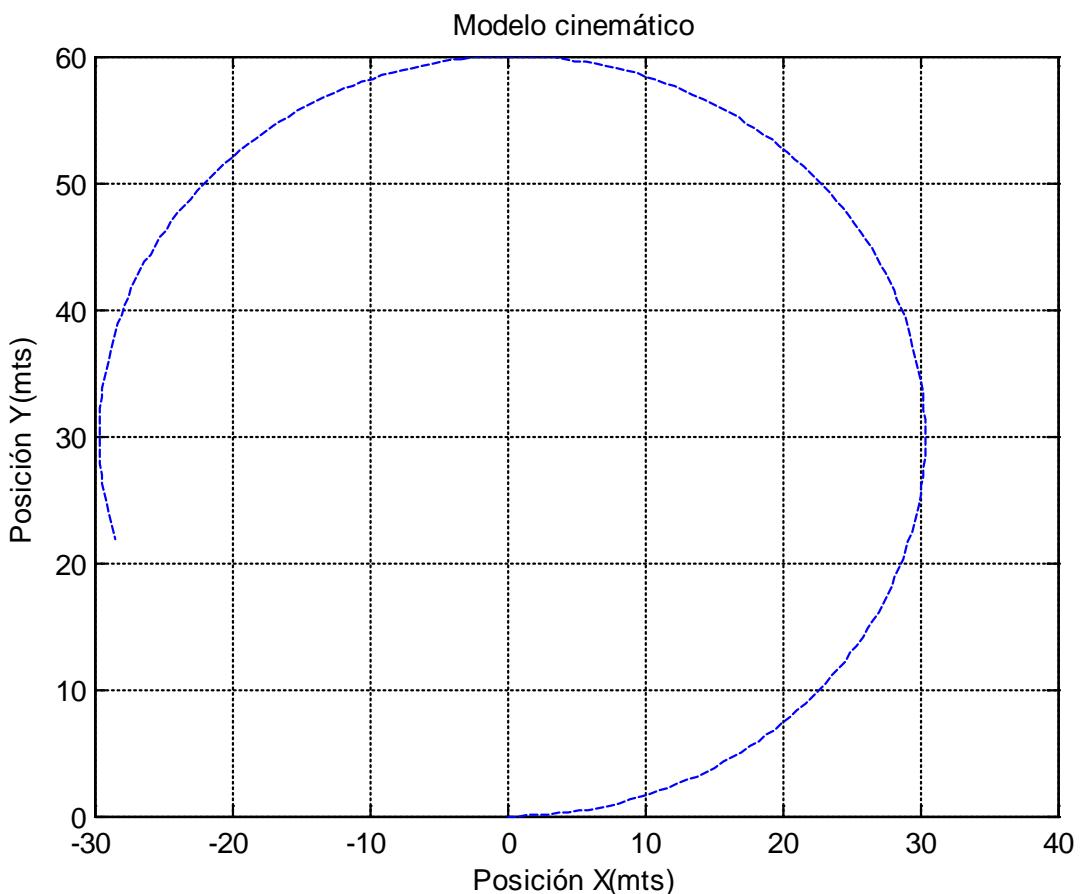
$$WR = 1 \frac{\text{rad}}{\text{s}}; WL = 2 \frac{\text{rad}}{\text{s}}$$

Para velocidades de diferentes magnitudes, la dirección de la trayectoria cambia.



Simulación con velocidades de magnitudes diferentes

$$WR = 2 \frac{\text{rad}}{\text{s}}; WL = 1 \frac{\text{rad}}{\text{s}}$$



Simulación con velocidades de magnitudes diferentes

3.2 Código del modelo cinemático

```
%%-----  
%---- DISEÑO Y CONTROL DE ROBOTS -----  
%-----  
%---- Modelo cinemático diferencial -----  
%-----  
%---- Lucas Rosas, céd.: 8-744-1135  
%---- Rangel Alvarado, céd.: 8-734-1444  
%---- Joaquín Valencia, céd.: 8-835-1792  
%---- Miguel Fuentes, céd.: E-8-98810  
%---- Carlos Samaniego, céd.: 6-714-698  
%% Definición de constantes  
r = 1; % radio de las ruedas  
c = .1; % separación entre las ruedas  
%% Tiempo  
dt=0.5;  
t = 0:dt:100; % Vector de tiempo
```

```

%% Velocidades de las llantas
WR=1; % Velocidad de las ruedas derechas
WL=1; % Velocidad de las ruedas izquierdas
%% Posición
Xi=0; % Posición inicial en X
Yi=0; % Posición inicial en Y
Xf = Xi; % Posición final en X
Yf = Yi; % Posición final en Y
X = Xf; % Vector de posiciones en X
Y = Yf; % Vector de posiciones en Y
%% Ángulos de dirección
thetai = 0; % Ángulo de dirección inicial
thetaf = thetai; % Ángulo de dirección final
theta = [];% Vector de ángulos de dirección

%% Ecuaciones cinemáticas
for k = 1:length(t)-1

Xf = Xf + ((r*(WR+WL)/2)*(dt))*(round(cos(thetaf)*10000)/10000); % Cálculo de la posición instantánea
en X
Yf = Yf + ((r*(WR+WL)/2)*(dt))*(round(sin(thetaf)*10000)/10000); % Cálculo de la posición instantánea
en Y
thetaf = thetaf + ((r*(WR-WL))/2*c*(dt)); % Cálculo del ángulo de dirección instantáneo theta

X = [X Xf]; % Generación del vector de posición en X
Y = [Y Yf]; % Generación del vector de posición en Y

theta = [theta thetaf]; % Generación del vector de ángulos de dirección
end
%% Gráficos
plot(X,Y,'--')
title('Modelo cinemático')
xlabel('Posición X(mts)')
ylabel('Posición Y(mts)')
grid on

```

4 Modelo dinámico

Para obtener el modelo dinámico del robot se deben aplicar y analizar las fuerzas sobre el cuerpo y de igual manera se deben tomar en consideración los momentos alrededor de algún punto del móvil, en nuestro caso este punto es el centro de gravedad del móvil. Dado que el modelo es descrito como un sistema de tres grados de libertad que solo permite movimientos en las direcciones longitudinal y lateral, y de igual forma desplazamientos angulares; las ecuaciones de fuerzas y momentos pueden ser expresadas como a continuación:

$$\sum Fx = m(\dot{u} - vr)$$

$$\sum Fy = m(\dot{v} - ur)$$

$$\sum Mz = Iz\dot{r}$$

Las fuerzas que actúan sobre el robot son ejercidas por las llantas y son proporcionales al torque aplicado menos la cantidad de torque necesario para romper la inercia de las llantas y el motor. Podemos distinguir los torques como: torque lineal y torque angular, y podemos representarlos respectivamente como se muestra a continuación:

$$Fx = \frac{Tlin}{Rt}$$

$$Tang = Iz \dot{w} = Iz \frac{\dot{u}}{Rt}$$

$$Fx = \frac{Tap - Tang}{Rt}$$

$$Fx = \frac{RtTap - Iz\dot{u}}{Rt^2}$$

La ecuación anterior puede ser modificada como corresponde modificando la componente u de velocidad en la dirección longitudinal para cada lado del robot y de esta manera tener dos expresiones de fuerza representadas como Fxi y Fxd.

Desarrollando las expresiones de sumatoria de fuerzas se pueden obtener las ecuaciones dinámicas que describen el modelo propuesto como representación del robot Summit utilizado que se pueden observar como a continuación:

$$\dot{V}_x = \frac{Fxi + Fxd}{m} + Vy w$$

$$\dot{V}_y = \frac{Fxi + Fxd}{m} - Vx w$$

$$\dot{r} = \frac{LrFxd - LlFxi}{Iz}$$

5 Controlador de seguimiento de trayectoria de un robot diferencial

A continuación se presenta el desarrollo de un controlador de seguimiento de trayectorias hecho en MATLAB.

La configuración del robot móvil utilizado es la de un robot diferencial ya que esta aproximación es una de las más usadas para "aproximar" el modelo de un robot de cuatro ruedas tipo "skid".

A grandes rasgos, el controlador se ocupa de seguir la trayectoria ajustando la velocidad lineal y la velocidad angular del robot, manteniendo una distancia de seguimiento definida por el usuario. Para

lograr esto se simula el movimiento de un punto a lo largo de la trayectoria deseada y ese punto es el utilizado como el objetivo del robot en cada instante de tiempo.

El código se implementó a propósito como código convencional y no como un modelo de simulink para presentar un algoritmo general que puede ser portado a otros lenguajes o sistemas.

5.1 Archivos del controlador

Para poder ejecutar el código es necesario que todos los archivos se encuentren en carpetas dentro del directorio(s) actual(es) de MATLAB.

- **Seguimiento_linea.m:** Es el archivo principal del controlador.
- **limiter.m:** Función de ayuda para limitar un valor dentro de un rango.
- **angLimiter:** Función de ayuda para limitar un ángulo en el rango $-\pi$ a π .
- **timedcomet:** Función de ayuda para visualizar un animación del movimiento del robot para tener una retroalimentación visual de la velocidad del movimiento. Es una modificación de la función COMET predefinida de MATLAB.
- **Archivos de curvas:** Son archivos de variables de MATLAB que contienen las trayectorias de prueba. Suministramos algunas (**Circulo.mat**, **Curva1.mat**, **Curva2.mat**, **Espiral.mat** y **Ocho.mat**) pero el usuario puede crear sus propias curvas de prueba siguiendo el formato definido.

5.2 Explicación del controlador

A continuación explicaremos el controlador paso a paso para facilitar su comprensión, posteriormente mostraremos algunas simulaciones y casos interesantes y al final podrá ver el código completo.

Para resumir se omitirán los comentarios en esta parte del documento pero en el archivo .m y al final de este documento los podrá encontrar.

5.3 Inputs del usuario

En esta sección se introducen los parámetros del controlador.

Con la siguiente línea se carga la trayectoria que deseamos que siga el robot. Puede utilizar las curvas suministradas como punto de partida o crear nuevas.

```
File = 'Circulo.mat';
```

Para definir el tiempo de simulación modifique las siguientes variables:

```
dt = 1;
TIEMPO = 700;
VisMov = false;
duracion = 0.5;
```

- **dt:** Indica el diferencial de tiempo entre cada iteración del proceso.
- **TIEMPO:** Es el tiempo total (segundos) de la simulación.

- **VisMov:** Active (true) esta variable para visualizar una animación del movimiento del robot sobre las gráficas. Esto facilita la visualización de la velocidad pero se debe tener en cuenta que esto consume más recursos del equipo.
- **duracion:** Es la duración de la visualización de la animación. El cumplimiento de este tiempo depende del poder de procesamiento del equipo con que se ejecuta el script.

```
xi = 0;
yi = 130;
thetai = -pi;
```

- **xi:** Posición inicial del robot en X.
- **yi:** Posición inicial del robot en Y.
- **thetai:** Dirección inicial del robot. **thetai** debe ser definida en radianes y puede tomar valores negativos.

```
Li = 20;
```

- **Li:** Es el valor de seguimiento. El valor de seguimiento indica a qué distancia del punto objetivo en la trayectoria tratará de mantenerse el robot. Mientras menor sea su valor más cerca tratará de estar el robot y más sensible se volverá a curvas ceradas.

```
KPv1 = .8;
KPv2 = 0.5;
KPang1 = .3;
KLi = 0.9;
```

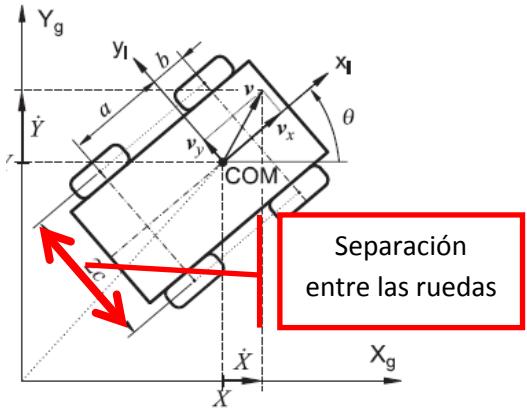
- **KPv1:** Constante proporcional de velocidad lineal.
- **KPv2:** Constante proporcional para el error.
- **KPang1:** Contante proporcional para la velocidad angular.
- **KLi:** Contante para el decrecimiento del valor de seguimiento.

```
Vlin_limited = true; % true/false
Vlin_max = 3;
Vlin_min = -3;
Ang_limited = true; % true/false
```

- **Vlin_limited:** Active o desactive si desea limitar la velocidad lineal del robot.
- **Vlin_max:** Valor máximo permitido para la velocidad lineal si se activa la limitación.
- **Vlin_min:** Valor mínimo permitido para la velocidad lineal si se activa la limitación.
- **Ang_limited:** Active o desactive si se desea limitar el rango de los ángulos al rango $-\pi$ a π .

```
Radio = 0.3;
Separacion = 0.5;
```

- **Radio:** Indica el radio de las ruedas del robot.
- **Separacion:** Indica la separación entre ruedas del robot.



Separación de entre la ruedas

5.4 Pre inicialización

En los siguientes bloques se realizan algunos procesos de pre inicialización y pre asignación de memoria.

```
load(File);
Trayectoria = A1;
TrayectoriaX = Trayectoria(1,:);
TrayectoriaY = Trayectoria(2,:);
```

Con las estas líneas se carga la trayectoria en el archivo definido por el usuario y se separan las coordenadas de la misma en dos vectores: **TrayectoriaX** y **TrayectoriaY**.

```
TIEMPO2 = 1:dt:TIEMPO;
```

Con esta línea se arma el vector de tiempo a utilizar. Nótese que el tamaño del vector dependerá del **TIEMPO** total de la simulación y el **dt** (diferencial de tiempo).

```
L = zeros(1, length(TIEMPO2));
xr = zeros(1, length(TIEMPO2));
yr = zeros(1, length(TIEMPO2));
thetar = zeros(1, length(TIEMPO2));
thetarGrad = zeros(1, length(TIEMPO2));
WL = zeros(1, length(TIEMPO2));
WR = zeros(1, length(TIEMPO2));
Wdiff = zeros(1, length(TIEMPO2));
Dist = zeros(1, length(TIEMPO2));
error = zeros(1, length(TIEMPO2));
integralerror = zeros(1, length(TIEMPO2));
Vlin = zeros(1, length(TIEMPO2));
Relang = zeros(1, length(TIEMPO2));
AngGiro = zeros(1, length(TIEMPO2));
```

```
Vang = zeros(1, length(TIEMPO2));
```

Con las líneas anteriores se pre asigna el espacio de memoria a utilizar por cada variable-vector para minimizar el tiempo de ejecución.

- **L:** Es el vector con los valores de seguimiento
- **xr:** Es el vector de las coordenadas en X del robot.
- **yr:** Es el vector de las coordenadas en Y del robot.
- **thetar:** Es el vector de las direcciones del robot en rad.
- **thetarGrad:** Es el vector de las direcciones del robot en grados.
- **WL:** Es el vector con las velocidades de la llanta izquierda del robot.
- **WR:** Es el vector con las velocidades de la llanta derecha del robot.
- **Wdiff:** Es el vector con las diferencias de las velocidades entre ambas llantas.
- **Dist:** Es el vector con las distancias al punto objetivo.
- **error:** Es el vector con los errores con respecto al objetivo.
- **integralerror:** Es el vector con las integrales del error.
- **Vlin:** Es el vector con las velocidades lineales del robot.
- **Relang:** Es el vector con las direcciones entre el centro robot y el punto objetivo.
- **AngGiro:** Es el vector con la diferencia entre **Relang** y la dirección del robot.
- **Vang:** Es el vector con las velocidades angulares del robot.

5.5 Inicialización de variables

En este bloque se inicializan algunas variables dependiendo del input del usuario para el tiempo t=0.

```
L(1) = Li;  
xr(1)= xi;  
yr(1)= yi;  
if Ang_limited == true  
    thetarai = angLimiter(thetai);  
end  
thetar(1) = thetarai;  
thetarGrad(1) = thetar(1)*180/pi;  
integralerror(1) = 0;  
WL(1) = 0;  
WR(1) = 0;  
Wdiff(1) = 0;
```

Lo que se debe resaltar en este bloque de código es el uso de la función **angLimiter()**. Esta función tiene el propósito de buscar el valor del ángulo correspondiente al rango $-\pi$ a π si la limitación es activada por el usuario. De esta forma un ángulo de 270° se traducirá a uno de -90° . De igual forma un ángulo de 450° se traducirá a un ángulo de 90° .

5.6 Bucle del controlador

Primero se define la cantidad de iteraciones de acuerdo a las variables de tiempo del usuario.

```
for i = 1:1:length(TIEMPO2)
```

Luego busco mi punto objetivo desplazándome por la trayectoria deseada en cada paso. Si no he llegado al final de la trayectoria tomo el punto actual como mi punto objetivo, en cambio, si me encuentro en el último punto y la simulación no ha terminado entonces sigo tomando ese como mi objetivo.

```
if i > length(Trayectoria)
    xgoal = TrayectoriaX(length(Trayectoria));
    ygoal = TrayectoriaY(length(Trayectoria));
    L(i) = L(i)*KLi;
else
    xgoal = TrayectoriaX(i);
    ygoal = TrayectoriaY(i);
end
```

En este bloque de código cabe señalar que introduce la constante de decrecimiento del seguimiento. Esto es importante ya que sin ella el robot nunca llegaría al final de la trayectoria, sino que se quedaría a una distancia **L** (entrada **Li**) del último punto. Lo que hace esta constante es disminuir el valor si se ha llegado al final de la trayectoria para que el robot siga avanzando.

```
Dist(i) = sqrt((xgoal - xr(i)).^2 + (ygoal - yr(i)).^2);
```

Con esta línea de código se calcula la distancia que hay entre el centro del robot y el punto objetivo de la trayectoria (teorema de Pitágoras).

```
error(i) = Dist(i)-L(i);
```

Con esta línea de código se calcula el error de posición. Este error es definido como la diferencia entre la distancia **Dist** y el valor de seguimiento **Li**.

```
if (i==1)
    integralerror(i) = 0;
else
    integralerror(i) = KPv2*(integralerror(i-1) + ((error(i)-error(i-1))*dt/2));
end
```

Con este bloque se calcula la integral del error. En este punto se utiliza la constante **KPv2** para definir el peso que tendrá el valor en el cálculo de la velocidad lineal.

```
Vlin(i) = KPv1*error(i) + integralerror(i);
```

Con esta línea se calcula la velocidad lineal del robot. La constante **KPv1** se utiliza para sintonizar la sensibilidad del controlador con respecto al error de posición.

```
if Vlin_limited == true  
    Vlin(i) = limiter(Vlin(i), Vlin_min, Vlin_max);  
end
```

Si la limitación de velocidad está activada entonces se verifica el valor de la velocidad lineal y se hace el ajuste correspondiente. En algunos casos es necesaria la limitación de velocidad para evitar que el controlador se desestabilice.

```
Relang(i) = atan2((ygoal - yr(i)),(xgoal - xr(i)));  
AngGiro(i) = Relang(i)-thetar(i);  
if Ang_limited == true  
    AngGiro(i) = angLimiter(AngGiro(i));  
end
```

Relang es el ángulo de la línea que se forma entre el centro del robot y el punto objetivo. **AngGiro** representa la diferencia entre **Relang** y la dirección del robot, en otras palabras, nos indica cuánto debe girar el robot para orientarse en la dirección del punto objetivo. Nuevamente, como se están realizando operaciones con ángulos que pueden dar respuestas fuera del rango $-\pi$ a π es necesario hacer el arreglo correspondiente.

```
Vang(i) = KPang1*AngGiro(i);
```

Esta línea calcula la velocidad angular del robot necesaria para orientarse. Se utiliza la constante **KPang1** para sintonizar la sensibilidad del controlador.

Luego de tener la velocidad lineal y angular del robot entonces puedo calcular los valores de la próxima iteración.

```
L(i+1) = L(i);
```

Con esta línea asigno el valor de seguimiento del próximo paso con el mismo valor del actual.

```
thetar(i+1) = thetar(i)+Vang(i)*dt;  
if Ang_limited == true  
    thetar(i+1) = angLimiter(thetar(i+1));  
end  
thetarGrad(i+1) = thetar(i+1)*180/pi;
```

Con estas líneas calculo la dirección que tendrá el robot en la próxima iteración utilizando la fórmula:

$$\theta_{siguiente} = \theta_{actual} + velocidad\ angular * tiempo$$

Luego se hace el arreglo correspondiente para que quede dentro del rango $-\pi$ a π y se transforma a grados para graficar.

Acto seguido debemos calcular las coordenadas en X y Y de la nueva posición.

```
xr(i+1) = xr(i) + Vlin(i)*(round(cos(theta(i+1))*10000)/10000)*dt;  
yr(i+1) = yr(i) + Vlin(i)*(round(sin(theta(i+1))*10000)/10000)*dt;
```

Para hacer esto utilizamos las siguientes fórmulas:

$$X_{siguiente} = X_{actual} + Vlineal * \cos \theta * t$$

$$Y_{siguiente} = Y_{actual} + Vlineal * \sin \theta * t$$

Luego, utilizamos el modelo dinámico del robot móvil diferencial para calcular las velocidades angulares individuales de cada rueda. Para hacer esto se resuelve el sistema de ecuaciones simultáneas que definen el modelo:

$$Vlineal = Radio\ de\ la\ rueda * \frac{(Velocidad\ angular\ izq + Velocidad\ angular\ der)}{2}$$

$$Vangular = Radio\ de\ la\ rueda * \frac{(-Velocidad\ angular\ izq + Velocidad\ angular\ der)}{Separación\ entre\ las\ ruedas}$$

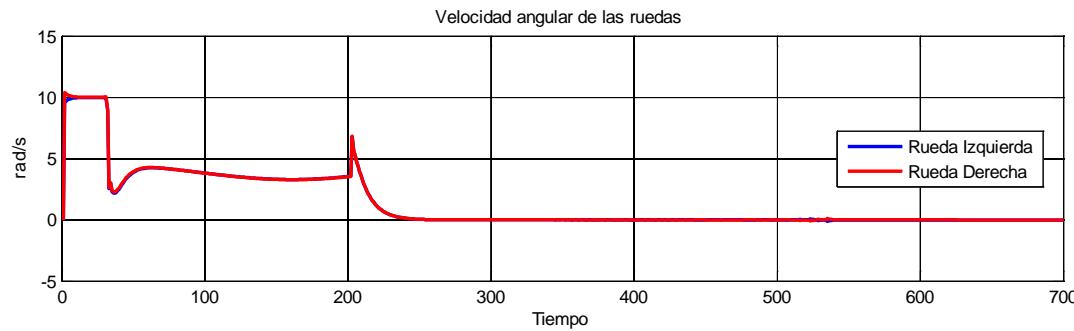
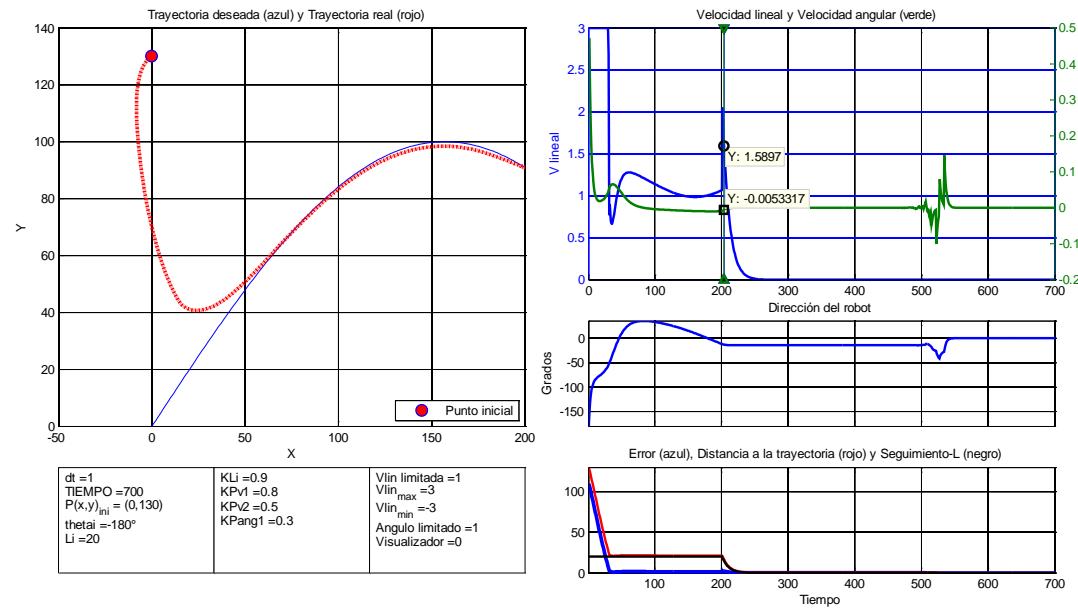
```
WL(i+1) = (2*Vlin(i)-Vang(i)*Separacion)/(2*Radio);  
WR(i+1) = WL(i+1) + (Vang(i)*Separacion)/Radio
```

Las ecuaciones en el código representan la resolución del sistema de ecuaciones.
El resto del código es utilizado solo para mostrar las gráficas.

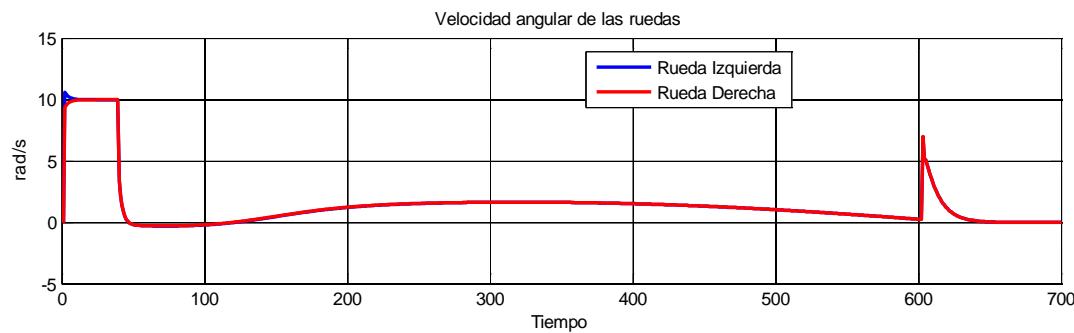
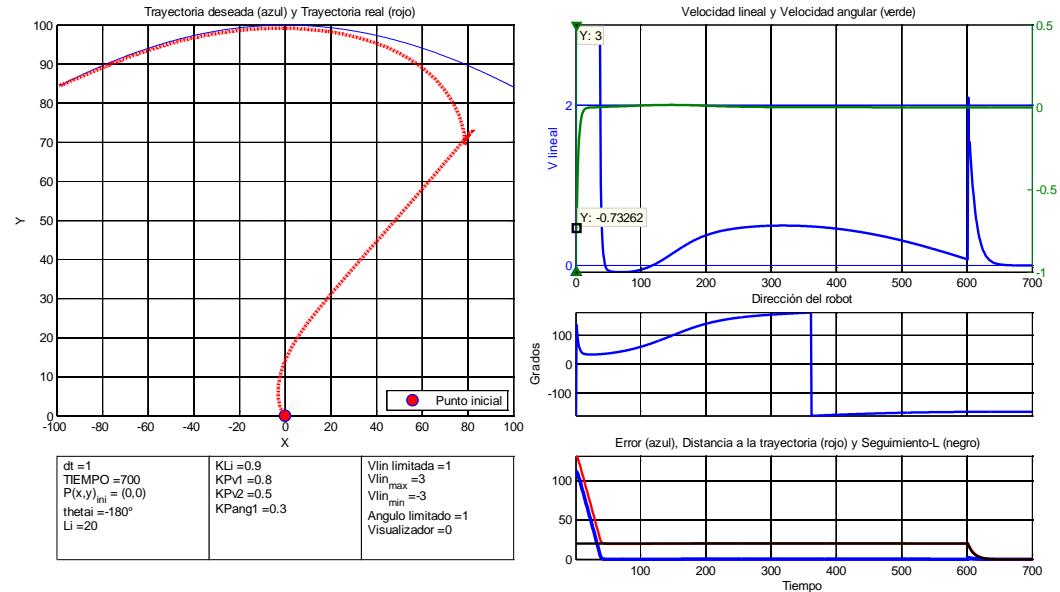
5.7 Simulaciones

A continuación se muestran algunas simulaciones hechas con los archivos de trayectorias suministrados y algunos casos especiales. En las imágenes se pueden observar los inputs del usuario para cada caso.

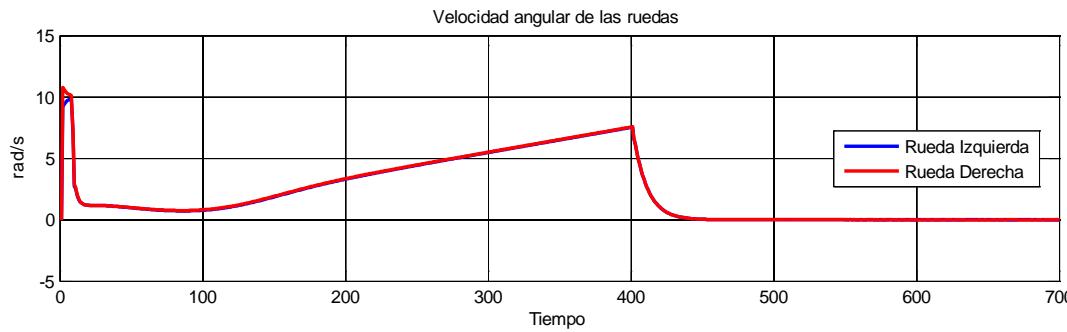
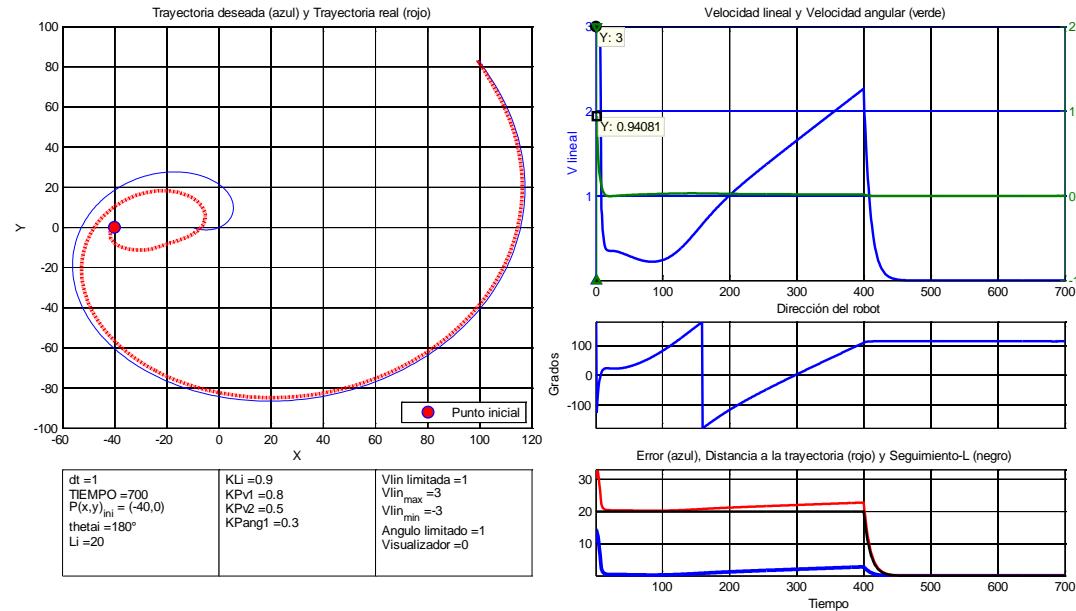
5.7.1 Seguimiento a Curva1.mat



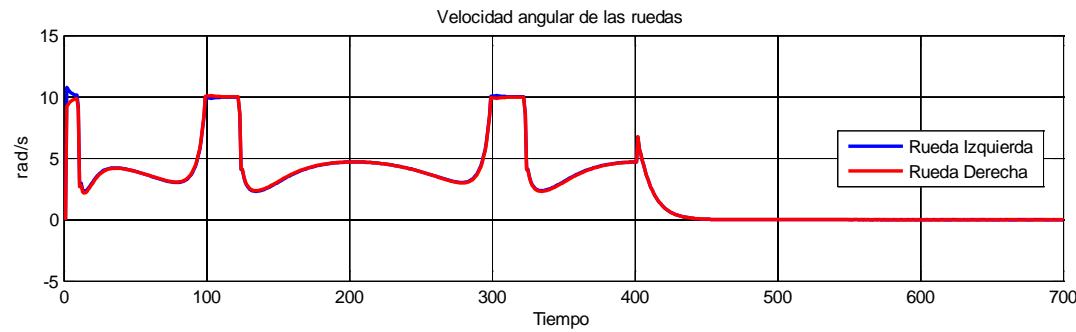
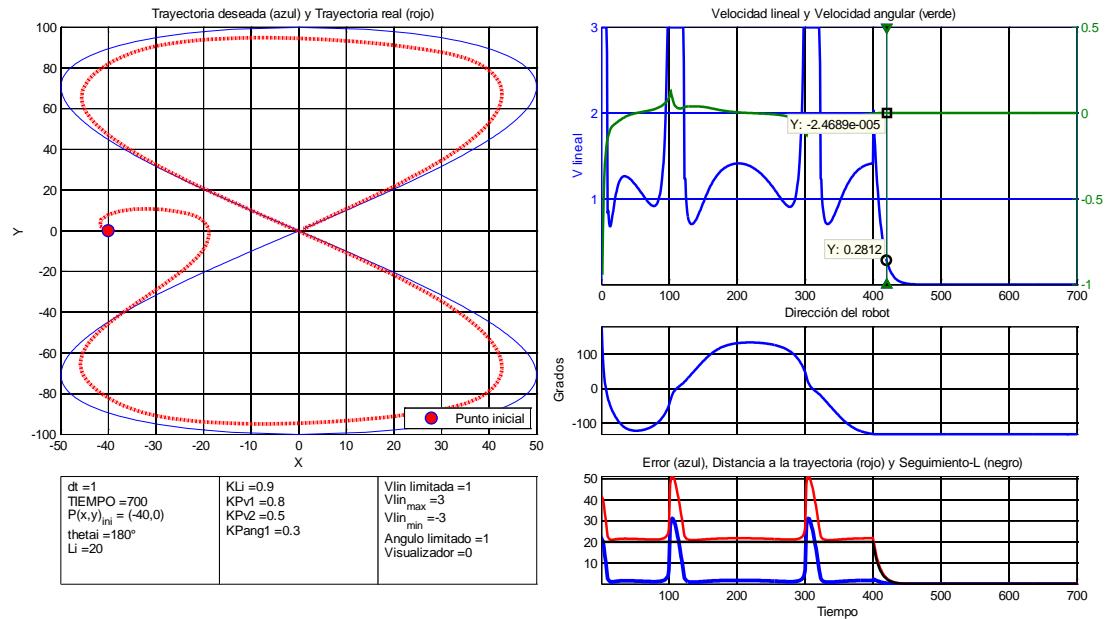
5.7.2 Seguimiento a Curva2.mat



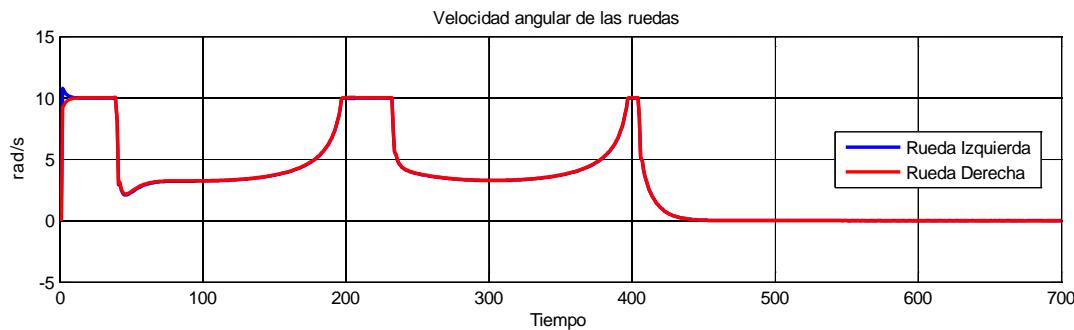
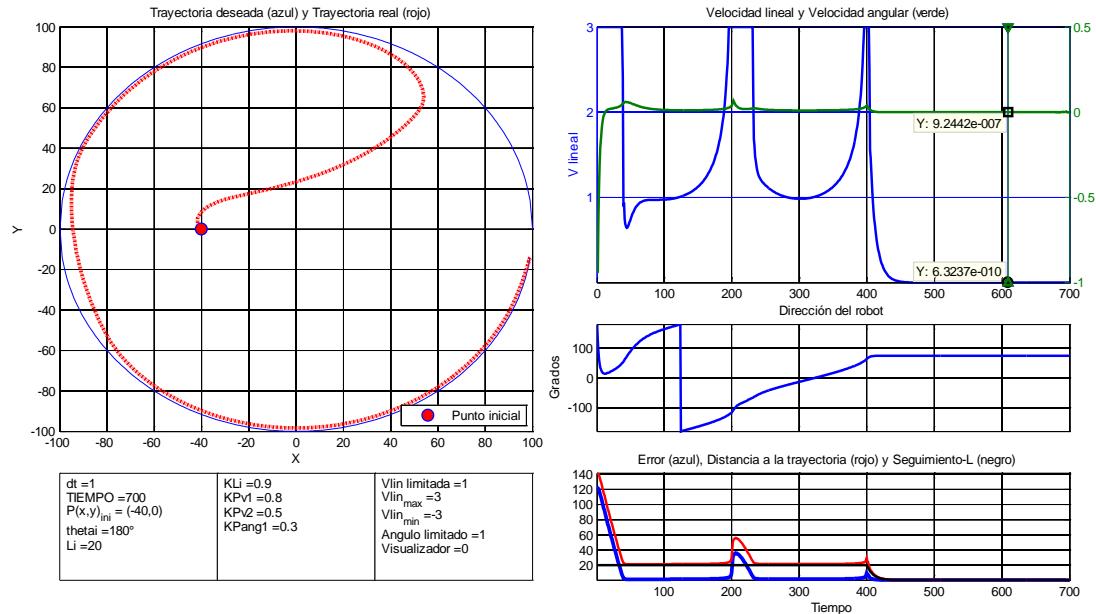
5.7.3 Seguimiento a Espiral.mat



5.7.4 Seguimiento a Ocho.mat

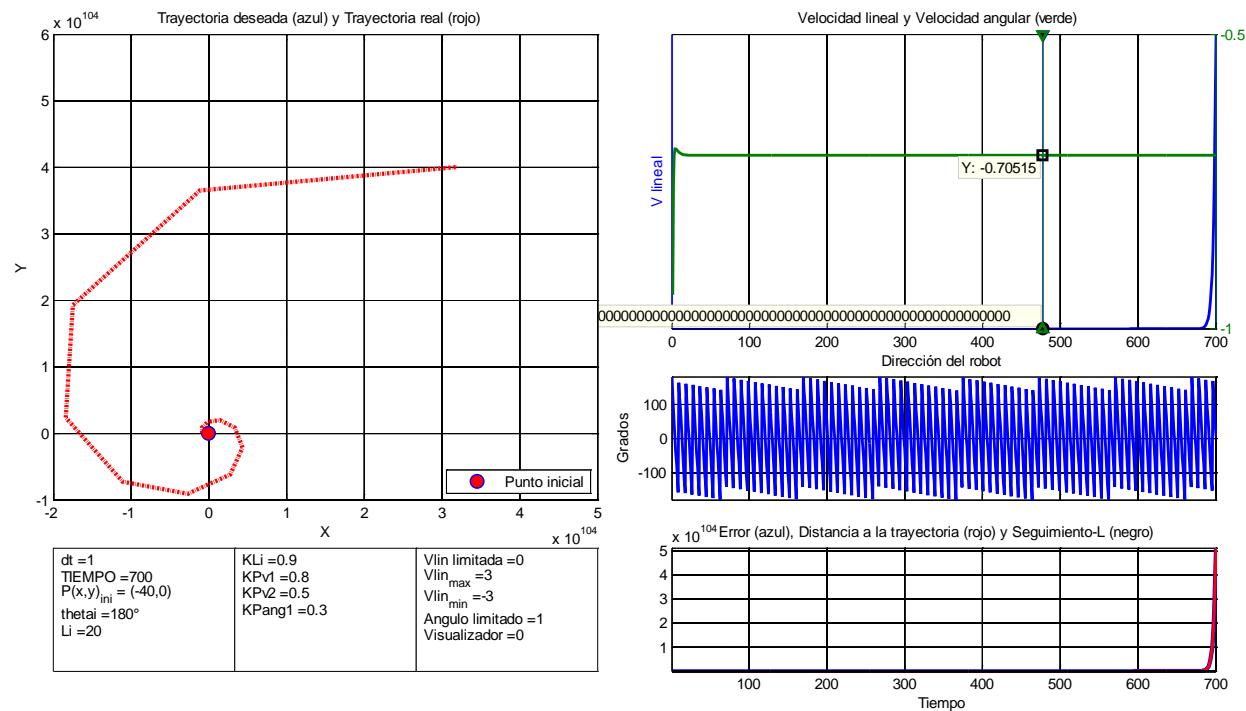


5.7.5 Seguimiento a Circulo.mat



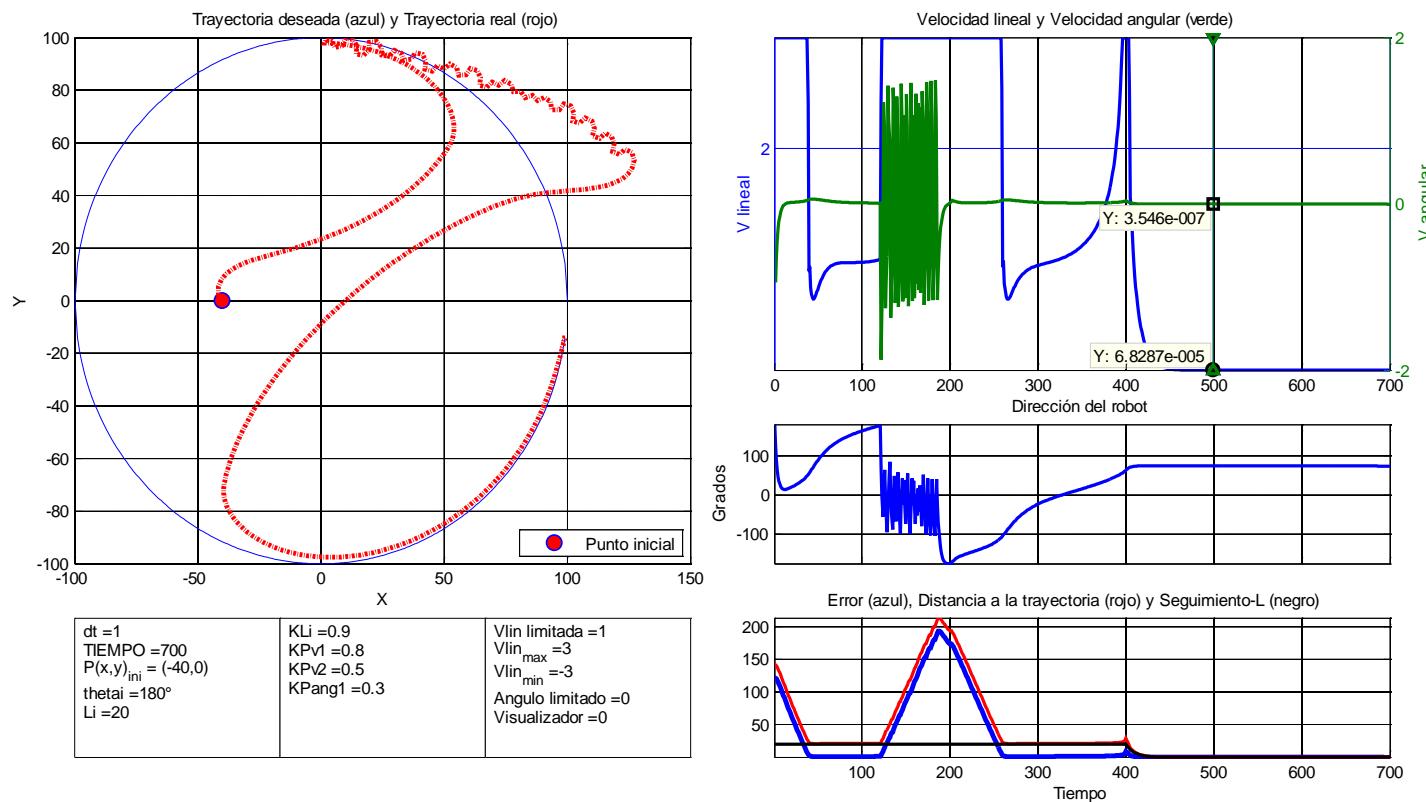
5.8 Efecto de la limitación de velocidad

En las simulaciones anteriores la velocidad se encuentra limitada, es por esto que la velocidad lineal alcanza un tope máximo (y/o mínimo) y el robot demora un tiempo en ajustarse a la trayectoria. Sin embargo, si eliminamos la limitación, en algunos casos el robot se descontrola y se pierde el seguimiento. En el siguiente ejemplo se está simulando con el archivo Circulo.mat bajo los mismos parámetros que en el punto anterior, sin embargo se ha eliminado la restricción a la velocidad lineal.



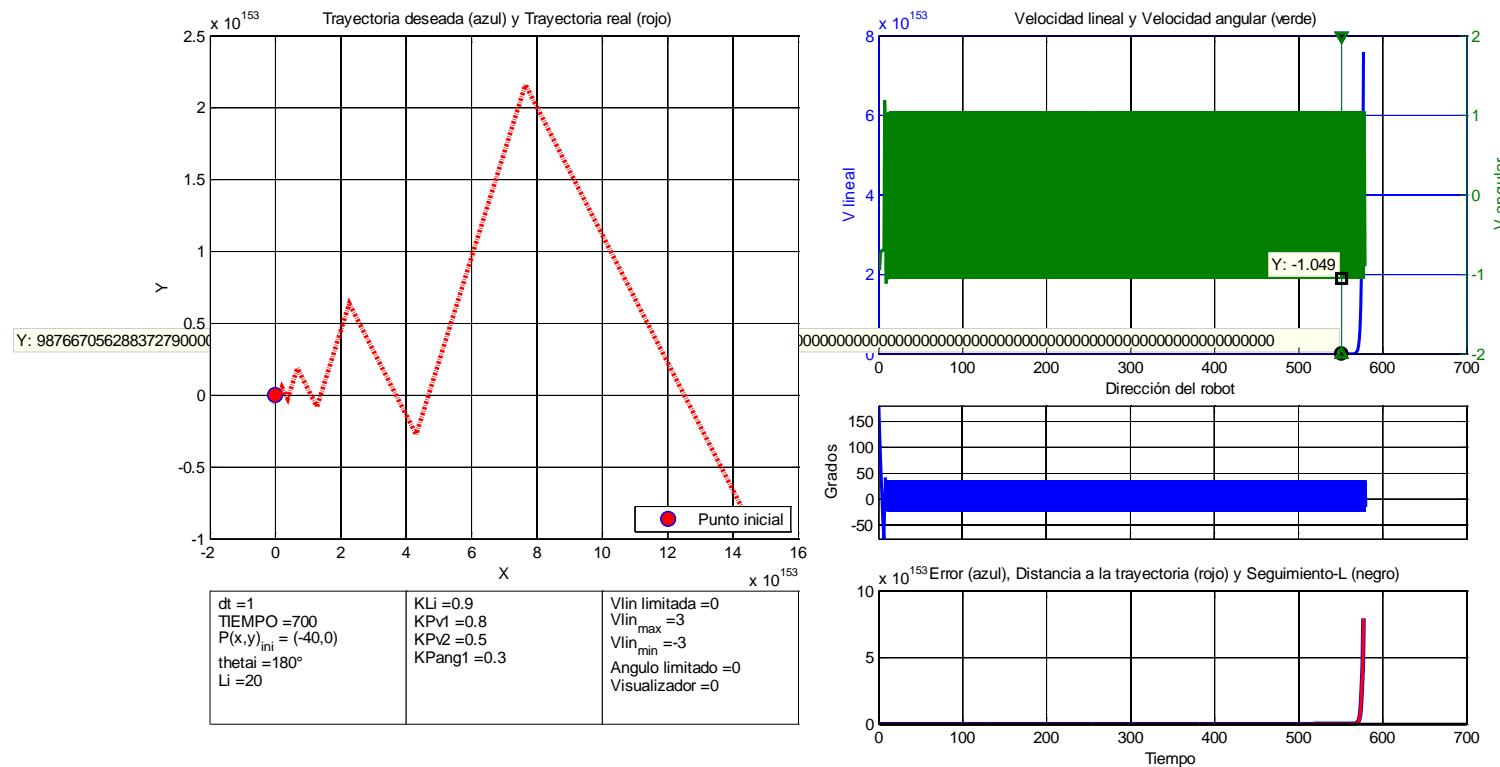
5.9 Efecto de la limitación de los ángulos al rango $-\pi \leq \theta \leq \pi$

En esta simulación se utilizan los mismos parámetros que en la anterior pero no se limitan los ángulos al rango mencionado. A pesar de que se está utilizando la función **atan2()** para calcular los ángulos también se hacen ciertas operaciones con ellos y los resultados pueden salirse del rango generando errores en la trayectoria.



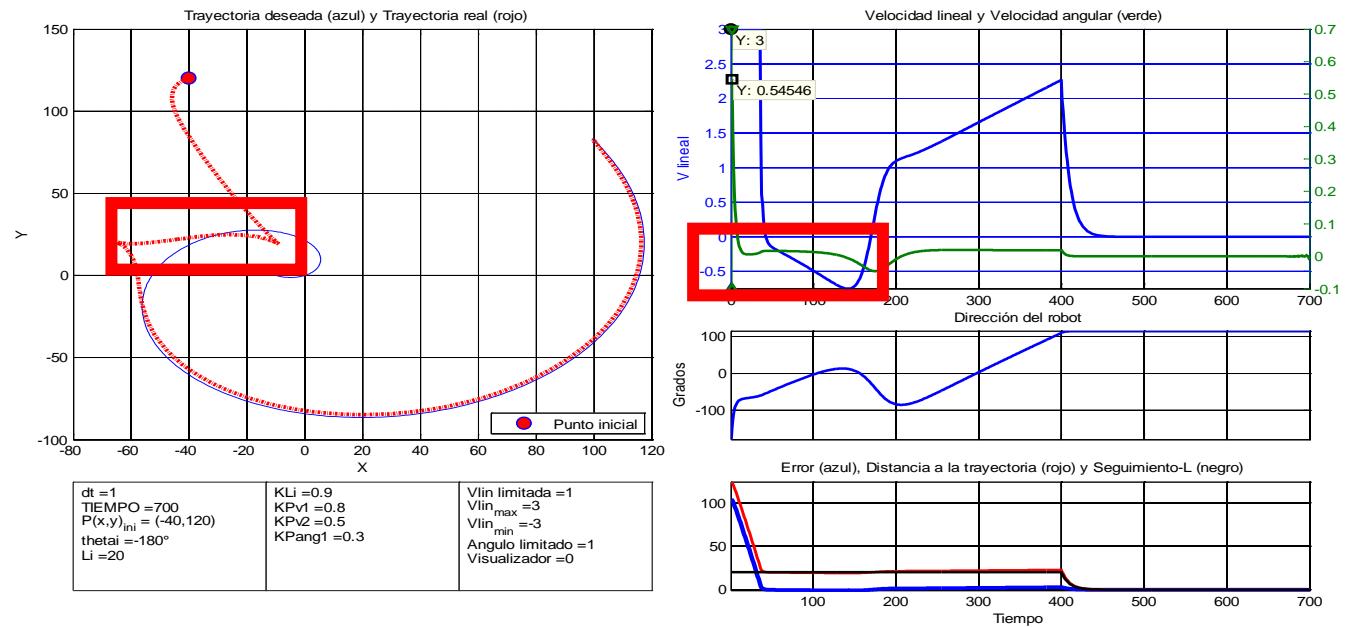
5.10 El peor de los casos (sin ningún tipo de limitación)

El "peor escenario" se da cuando no se limitan, ni la velocidad, ni el rango de los ángulos. En la siguiente simulación se utilizan los mismo parámetros que en las dos anteriores.

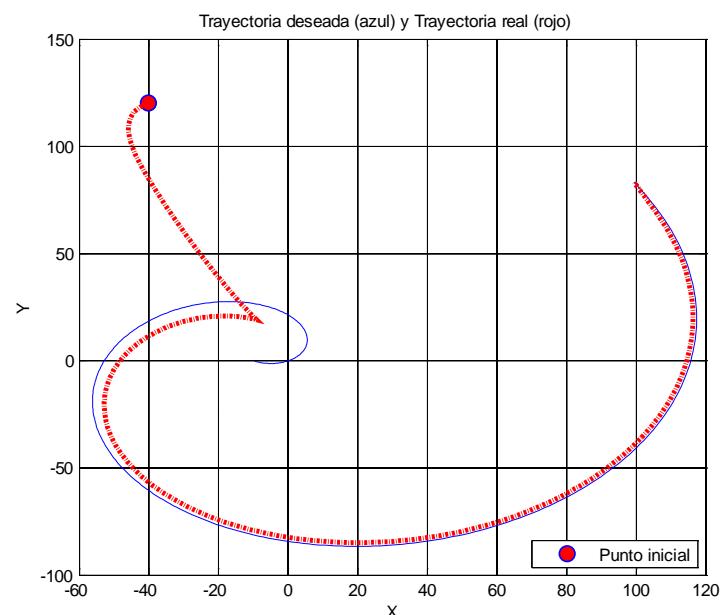


5.11 Restricción de velocidad a valores positivos

En las simulaciones hay ocasiones en donde el robot avanza hacia atrás para acomodarse y seguir la trayectoria. Podemos ver en la siguiente simulación que durante un período el controlador genera una velocidad lineal negativa.



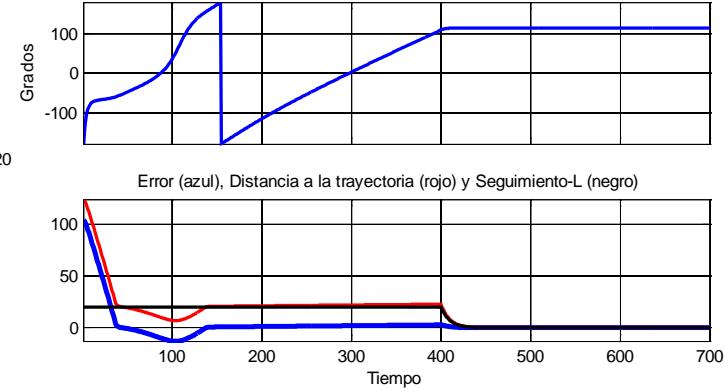
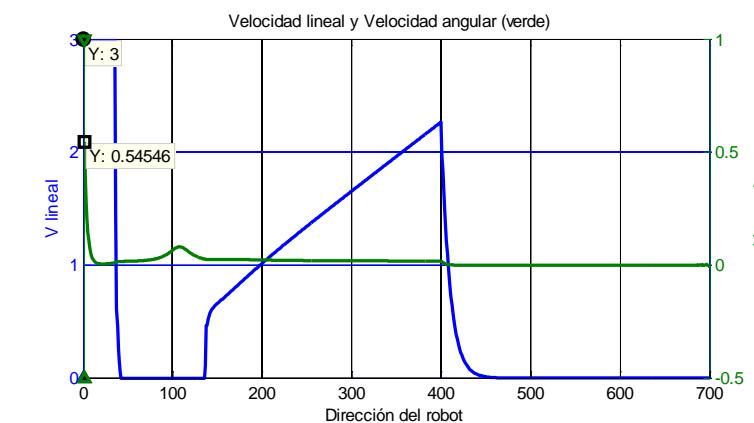
Si corremos la misma simulación, pero restringimos la velocidad lineal a valores positivos obtenemos un seguimiento más coherente:



dt =1
TIEMPO =700
 $P(x,y)_{ini} = (-40,120)$
thetai =-180°
Li =20

KLi =0.9
KPv1 =0.8
KPv2 =0.5
KPang1=0.3

Vlin limitada =1
 $Vlin_{max} =3$
 $Vlin_{min} =0$
Angulo limitado =1
Visualizador =0



5.12 Códigos completos

Seguimiento_linea.m

```
%%-----  
%---- DISEÑO Y CONTROL DE ROBOTS -----  
%-----  
%---- Controlador de robot diferencial -----  
%---- para seguimiento de trayectorias -----  
%-----  
%---- Lucas Rosas, céd.: 8-744-1135  
%---- Rangel Alvarado, céd.: 8-734-1444  
%---- Joaquín Valencia, céd.:8-835-1792  
%---- Miguel Fuentes, céd.: E-8-98810  
%---- Carlos Samaniego, céd.: 6-714-698
```

```
clear  
clc
```

```
%% INPUTS DEL USUARIO -----  
% -----
```

```
% >TRAYECTORIA OBJETIVO:  
% -->Indico el archivo externo que  
% -->tiene los puntos de la trayectoria  
% -->(el archivo debe estar en el current path  
% --> de MATLAB)  
File = 'Espiral.mat';
```

```
% >TIEMPO:  
% -->Delta de tiempo  
dt = 1;  
% -->Tiempo total de la simulacion:  
TIEMPO = 700;  
% -->Activar visualizacion de movimiento:  
VisMov = false;  
% -->Duracion de visualizacion de movimiento:  
duracion = 0.5;
```

```
% >VALORES INICIALES DEL ROBOT:  
% -->Posicion inicial en X  
xi = -40;
```

```

% -->Posición inicial en Y
yi = 120;
% -->Dirección inicial
thetai = -pi;

% >VALOR DE SEGUIMIENTO INICIAL:
% -->Con esto defino a que distancia debe
% -->seguir el robot al punto objetivo
Li = 20;

% >CONSTANTES DE SIMULACION:
% -->Constante proporcional de velocidad lineal
KPv1 = .8;
% -->Constante proporcional para el error
KPv2 = 0.5;
% -->Constante proporcional de velocidad angular
KPang1 = .3;
% -->Constante de disminución de valor de seguimiento
KLi = 0.9;

% >RESTRICCIÓN DE VELOCIDAD LINEAL:
% -->Indico si deseo restringir la velocidad lineal
Vlin_limited = true; % true/false
% -->Límite superior
Vlin_max = 3;
% -->Límite inferior
Vlin_min = 0;

% >RESTRICCIÓN DE MAGNITUD ANGULAR:
% -->Indico si deseo restringir la dirección al rango
% -->-pi a pi
Ang_limited = true; % true/false

% >PARAMETROS FISICOS DEL ROBOT:
% -->Indico el radio de las llantas
Radio = 0.3;
% -->Indico la separación entre llantas opuestas
Separacion = 0.5;

%>FIN DE INPUTS DEL USUARIO

%% A PARTIR DE ESTE PUNTO INICIA EL CONTROLADOR
%% Obtengo los puntos de la trayectoria a seguir:

```

```

load(File);
Trayectoria = A1;
TrayectoriaX = Trayectoria(1,:);
TrayectoriaY = Trayectoria(2,:);

%% Armo el vector de Tiempo
TIEMPO2 = 1:dt:TIEMPO;

%% Pre asignación de memoria
% Esto es solo para acelerar la ejecución del programa
L = zeros(1, length(TIEMPO2));
xr = zeros(1, length(TIEMPO2));
yr = zeros(1, length(TIEMPO2));
thetar = zeros(1, length(TIEMPO2));
thetarGrad = zeros(1, length(TIEMPO2));
WL = zeros(1, length(TIEMPO2));
WR = zeros(1, length(TIEMPO2));
Wdiff = zeros(1, length(TIEMPO2));
Dist = zeros(1, length(TIEMPO2));
error = zeros(1, length(TIEMPO2));
integralerror = zeros(1, length(TIEMPO2));
Vlin = zeros(1, length(TIEMPO2));
Relang = zeros(1, length(TIEMPO2));
AngGiro = zeros(1, length(TIEMPO2));
Vang = zeros(1, length(TIEMPO2));

%% Inicialización
L(1) = Li; % vector de seguimiento
xr(1)= xi; % vector de posición en X
yr(1)= yi; % vector de posición en Y
% Fix para el ángulo inicial
% Debo mantener el ángulo en el rango -pi <= X <= pi
if Ang_limited == true
    thetarai = angLimiter(thetarai);
end
thetar(1) = thetarai; % vector de dirección en rad
thetarGrad(1) = thetar(1)*180/pi; % dirección en grados
integralerror(1) = 0; % integral del error
WL(1) = 0; % velocidad angular de ruedas izquierdas inicial
WR(1) = 0; % velocidad angular de ruedas derechas inicial
Wdiff(1) = 0; % diferencia entre las velocidades (para visualizar)

%% Inicio
for i = 1:1:length(TIEMPO2)

```

```

% Tomo el punto de la trayectoria que voy a seguir
if i > length(Trayectoria)
    xgoal = TrayectoriaX(length(Trayectoria));
    ygoal = TrayectoriaY(length(Trayectoria));
    L(i) = L(i)*KLi; % disminuyo el valor de seguimiento para llegar al final de la trayectoria
else
    xgoal = TrayectoriaX(i);
    ygoal = TrayectoriaY(i);
end

% Calculo la distancia a ese punto:
Dist(i) = sqrt((xgoal - xr(i)).^2 + (ygoal - yr(i)).^2);

% Calculo el error
error(i) = Dist(i)-L(i);

% Calculo la integral del error
if (i==1)
    integralerror(i) = 0;
else
    integralerror(i) = KPv2*(integralerror(i-1) + ((error(i)-error(i-1))*dt/2));
end

% Calculo la velocidad lineal:
Vlin(i) = KPv1*error(i) + integralerror(i);

% Restricción de velocidad lineal
if Vlin_limited == true
    Vlin(i) = limiter(Vlin(i), Vlin_min, Vlin_max);
end

% Calculo el ángulo entre el robot y el punto:
Relang(i) = atan2((ygoal - yr(i)),(xgoal - xr(i)));

% Cálculo la diferencia entre el ángulo del robot y el ángulo al punto:
AngGiro(i) = Relang(i)-theta(i);

% Fix para el ángulo
% Debo mantener el angulo en el rango -pi <= X <= pi
if Ang_limited == true
    AngGiro(i) = angLimiter(AngGiro(i));
end

```

```

% Calculo la velocidad angular:
Vang(i) = KPang1*AngGiro(i);

% CALCULO LA NUEVA POSICION Y PARAMETROS PARA EL SIGUIENTE PASO
if (i+1) <= length(TIEMPO2)
    % Calculo el nuevo valor de seguimiento:
    L(i+1) = L(i);
    % Calculo la nueva dirección del robot:
    thetar(i+1) = thetar(i)+Vang(i)*dt;
    % Fix para el ángulo
    % Debo mantener el ángulo en el rango -pi <= X <= pi
    if Ang_limited == true
        thetar(i+1) = angLimiter(thetar(i+1));
    end
    % Nueva dirección en grados:
    thetarGrad(i+1) = thetar(i+1)*180/pi;
    % Calculo la nueva posición:
    xr(i+1) = xr(i) + Vlin(i)*(round(cos(thetar(i+1)))*10000)/10000*dt;
    yr(i+1) = yr(i) + Vlin(i)*(round(sin(thetar(i+1)))*10000)/10000*dt;
    % Calculo las nuevas velocidades angulares de las ruedas:
    % Este cálculo proviene de la resolucion de las
    % ecuaciones simultaneas del modelo dinamico
    % Vlin = Radio*(WL+WR)/2
    % Vang = Radio*(-WL+WR)/Separacion
    WL(i+1) = (2*Vlin(i)-Vang(i)*Separacion)/(2*Radio);
    WR(i+1) = WL(i+1) + (Vang(i)*Separacion)/Radio;
    % Calculo la diferencia entre las velocidades
    % para poder visualizarlas mejor
    Wdiff(i+1) = WL(i+1)-WR(i+1);

end

%% INICIALIZO EL DESPLIEGUE DE LAS GRAFICAS
figure
set(gcf,'units','normalized','outerposition',[0 0 1 1]); % Maximizar figura
set(gcf, 'ToolBar', 'none');
set(gcf,'Color','white')
set(gca,'LooseInset',get(gca,'TightInset'));

%% DESPLIEGO LA VELOCIDAD LINEAL Y ANGULAR
subplot(4,4,[3 4 7 8])
[ax h1 h2] = plotyy(TIEMPO2,Vlin, TIEMPO2, Vang);

```

```

title('Velocidad lineal y Velocidad angular (verde)');
ylabel(ax(1), 'V lineal');
axis tight
set(gca,'LooseInset',get(gca,'TightInset'));
set(gca, 'XTickLabel', []);
ylabel(ax(2), 'V angular');
axis tight
set(gca,'LooseInset',get(gca,'TightInset'));
set(gca, 'XTickLabel', []);
set(h1,'Color','b')
set(h1,'LineWidth',2)
set(h2,'Color',[0 0.5 0])
set(h2,'LineWidth',2)

% Despliego cursores para ayudar a la visualización
% ya que las magnitudes pueden ser diferentes
hCursorbar = graphics.cursorbar(h1); drawnow
hCursorbar.CursorLineColor = 'blue';
hCursorbar.CursorLineWidth = .5;
hCursorbar.Orientation = 'vertical';
hCursorbar.TargetMarkerSize = 8;
hCursorbar.TargetMarkerStyle = 'o';

hCursorbar2 = graphics.cursorbar(h2); drawnow
hCursorbar2.CursorLineColor = [0 0.5 0];
hCursorbar2.CursorLineWidth = .5;
hCursorbar2.Orientation = 'vertical';
hCursorbar2.TargetMarkerSize = 8;
hCursorbar2.TargetMarkerStyle = 's';

grid on

%% DESPLIEGO LA DIRECCION DEL ROBOT
subplot(4,4,[11 12])
plot(thetaGrad,'blue','LineWidth',2);
set(gca, 'XTickLabel', []);
axis tight
set(gca,'LooseInset',get(gca,'TightInset'));
title('Dirección del robot');
ylabel('Grados');
grid on
hold off

%% DESPLIEGO EL ERROR, DIST y L
subplot(4,4,[15 16])

```

```

plot(error,'blue','LineWidth',3);
axis tight
set(gca,'LooseInset',get(gca,'TightInset'));
title('Error (azul), Distancia a la trayectoria (rojo) y Seguimiento-L (negro)');
xlabel('Tiempo');
grid on
hold on
plot(Dist,'red','LineWidth',2);
axis tight
hold on
plot(L,'black','LineWidth',2);
axis tight
hold off

%% DESPLIEGO LOS PARAMETROS DE LA SIMULACION
tdt = strcat('dt = ', num2str(dt));
tTIEMPO = strcat('TIEMPO = ', num2str(TIEMPO));
tposi = strcat('P(x,y)_i_n_i = (' , num2str(xi), ', ', num2str(yi), ')');
tthetai = strcat('thetai = ', num2str(thetai*180/pi), '°');
tLi = strcat('Li = ', num2str(Li));
tKLi = strcat('KLi = ', num2str(KLi));
tKPx1 = strcat('KPx1 = ', num2str(KPx1));
tKPx2 = strcat('KPx2 = ', num2str(KPx2));
tKPang1 = strcat('KPang1 = ', num2str(KPang1));
tVlin_limited = strcat('Vlin limitada = ', num2str(double(Vlin_limited)));
tVlin_max = strcat('Vlin_m_a_x = ', num2str(Vlin_max));
tVlin_min = strcat('Vlin_m_i_n = ', num2str(Vlin_min));
tAng_limited = strcat('Angulo limitado = ', num2str(double(Ang_limited)));
tVisMov = strcat('Visualizador = ', num2str(double(VisMov)));
str1 = {tdt, tTIEMPO, tposi, tthetai, tLi};
str2 = {tKLi, tKPx1, tKPx2, tKPang1};
str3 = {tVlin_limited, tVlin_max, tVlin_min, tAng_limited, tVisMov};

subplot(4,4,[13 14])
set(gca,'Color','none');
set(gca,'Visible','off');
pos = get(gca,'position');
annotation('textbox', 'String', str1, 'BackgroundColor', 'white', 'EdgeColor', 'black', 'Position', [pos(1) pos(2)
pos(3)/3 pos(4)], 'FitBoxToText', 'off');
annotation('textbox', 'String', str2, 'BackgroundColor', 'white', 'EdgeColor', 'black', 'Position',
[pos(1)+pos(3)/3 pos(2) pos(3)/3 pos(4)], 'FitBoxToText', 'off');
annotation('textbox', 'String', str3, 'BackgroundColor', 'white', 'EdgeColor', 'black', 'Position',
[pos(1)+2*pos(3)/3 pos(2) pos(3)/3 pos(4)], 'FitBoxToText', 'off');

%% DESPLIEGO LA RUTA DESEADA Y LA RUTA REAL DEL ROBOT

```

```

subplot(4,4,[1 2 5 6 9 10])
plot(TrayectoriaX,TrayectoriaY,'blue');
set(gca,'LooseInset',get(gca,'TightInset'));
title('Trayectoria deseada (azul) y Trayectoria real (rojo)');
xlabel('X');
ylabel('Y');
grid on
hold on
point = plot(xr(1),yr(1),'o', 'MarkerSize',10, 'MarkerFaceColor','red', 'MarkerEdgeColor','blue', 'LineWidth',1);
legend(point,'Punto inicial', 'Location', 'SouthEast');
hold on
plot(xr,yr,'red','LineWidth',3,'LineStyle','-.');
hold on
% Si la visualizacion de movimiento esta activada
% muestro el comet de la trayectoria
if VisMov == true
    timedcomet(xr,yr,0,duracion/length(TIEMPO2));
end
hold off

%% DESPLIEGO LAS VELOCIDADES ANGULARES DE LAS RUEDAS DEL ROBOT
figure
set(gcf,'units','normalized','outerposition',[0.2 0 .6 0.4]); % tamaño de figura
set(gcf, 'ToolBar', 'none');
set(gcf,'Color','white')
set(gca,'LooseInset',get(gca,'TightInset'));
IZQ = plot(TIEMPO2, WL, 'blue', 'LineWidth', 2);
set(gca,'LooseInset',get(gca,'TightInset'));
title('Velocidad angular de las ruedas');
xlabel('Tiempo');
ylabel('rad/s');
grid on
hold on
DER = plot(TIEMPO2, WR, 'red', 'LineWidth', 2);
legend('Rueda Izquierda', 'Rueda Derecha', 'Location', 'Best');

hold off

%% BORRO TODAS LAS VARIABLES
% Esto se hace para liberar memoria
% Si desea analizar las variables generadas
% comente la siguiente linea:
Clear

```

limiter.m

```

function [ new_value ] = limiter( value, min, max )
%LIMITER Limitador de valores
% LIMITER(VALUE, MIN, MAX): limita un valor de entrada (VALUE)
% entre dos valores (MIN) y (MAX). Si el valor de entrada es
% menor que MIN devuelve MIN y si es mayor que MAX devuelve MAX.

if value > max
    new_value = max;
elseif value < min
    new_value = min;
else
    new_value = value;
end

end

```

angLimiter.m

```

function [ new_value ] = angLimiter( value )
%ANGLIMITER Limitador de valores
% ANGLIMITER(VALUE): transforma un angulo de entrada (VALUE)
% en radianes al rango -pi < VALUE < pi.

if (value < -2*pi) || (value > 2*pi)
    value = mod(value,2*pi);
end

if value < -pi
    new_value = value + 2*pi;
elseif value > pi
    new_value = value - 2*pi;
else
    new_value = value;
end

end

```

timedcomet.m

Esta función no se incluye en este documento puesto que sólo es una pequeña modificación a la función **COMET()** de MATLAB. Si el usuario desea puede revisar la función directamente dentro de los archivos suministrados.

6 Prueba remota desde una terminal telnet con el robot summit XL

Para probar el robot de forma remota se recomienda antes verificar todos los manuales, sin embargo, con conocimiento previo de la plataforma puede pasar a ver el pdf “2 SUMMIT XL – System_Installation_and_Configuration_Manual.pdf”.

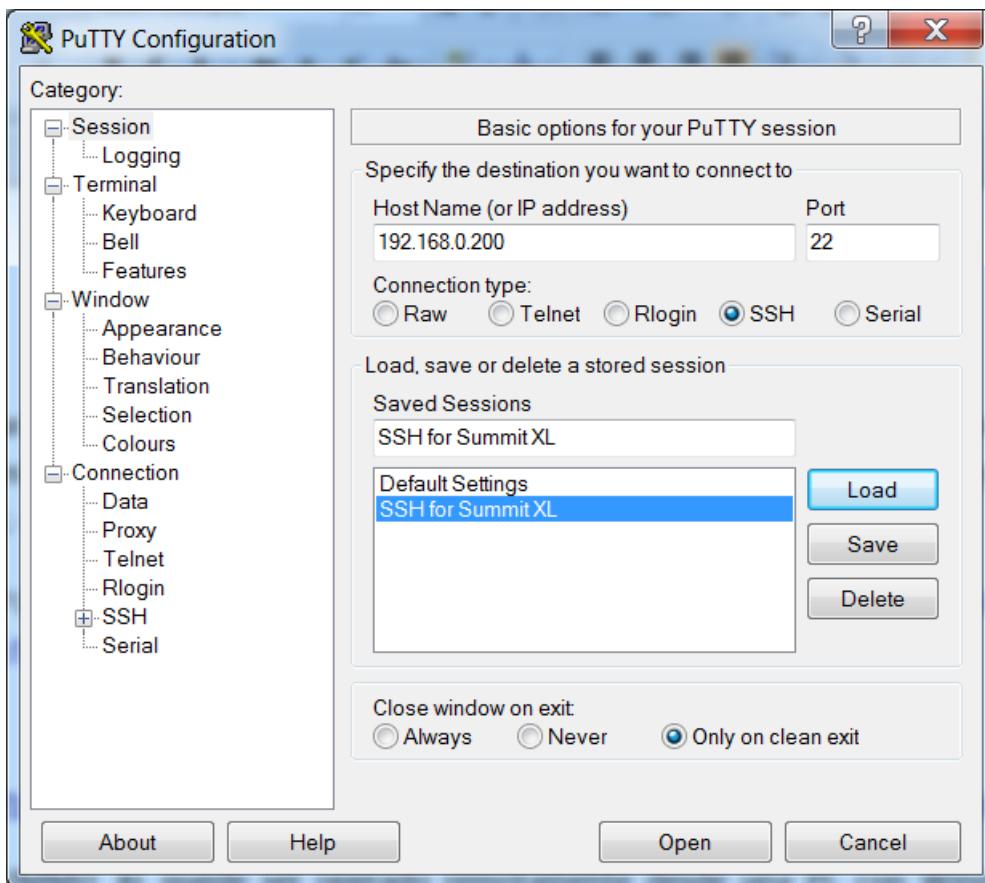
Se puede acceder a la información en línea de la página: <https://code.google.com/p/summit-xl-ros-stack/>, en donde está el repositorio del STACK del robot.

O en la siguiente dirección, la cual puede encontrar preguntas y respuestas comunes al robot.

<http://answers.ros.org/question/49546/how-to-install-robotnik-summit-xl-stack/>

SUMMIT XL puede ser operado remotamente desde una PC con Windows o Linux se debe tener como mínimo una terminal cliente de SSH, por consiguiente se debe bajar putty de la página principal.

Seguido configurar putty como sigue en la terminal:



El nombre de usuario, y del robot son summit sumit respectivamente.

A screenshot of a terminal window titled "summit@summit: ~". The window shows a standard Linux login sequence:

```
summit@summit: ~
login as: summit
summit@192.168.0.200's password:
Welcome to Ubuntu 12.04.1 LTS (GNU/Linux 3.2.0-32-generic x86_64)

 * Documentation: https://help.ubuntu.com/

Last login: Thu May  2 04:14:24 2013 from 192.168.0.2
ROBOTNIK SUMMIT XL
summit@summit:~$
```

Utilizar

A screenshot of a terminal window titled "summit@summit: /". It shows the user navigating to the root directory and listing files, then editing the "/etc/hosts" file.

```
summit@summit:/$ ls
bin  dev  initrd.img   lib64   mnt   root  selinux  tmp  vmlinuz
boot  etc  initrd.img.old  lost+found  opt   run   srv   usr  vmlinuz.old
cdrom  home  lib       media   proc   sbin  sys   var

summit@summit:/$ vi /etc/hosts
127.0.0.1      localhost
127.0.1.1      summit
#192.168.0.200  summit
192.168.0.3      trurl

# The following lines are desirable for IPv6 capable hosts
::1      ip6-localhost ip6-loopback
fe00::0  ip6-localnet
ff00::0  ip6-mcastprefix
ff02::1  ip6-allnodes
ff02::2  ip6-allrouters
~
```

No se puede instalar ROS Electric en Oneiric Ocelot, por consiguiente se decidió instalar ROS Fuerte la cual es la última versión de ROS soportada en Summit XL hasta el mes de Mayo del 2013¹.

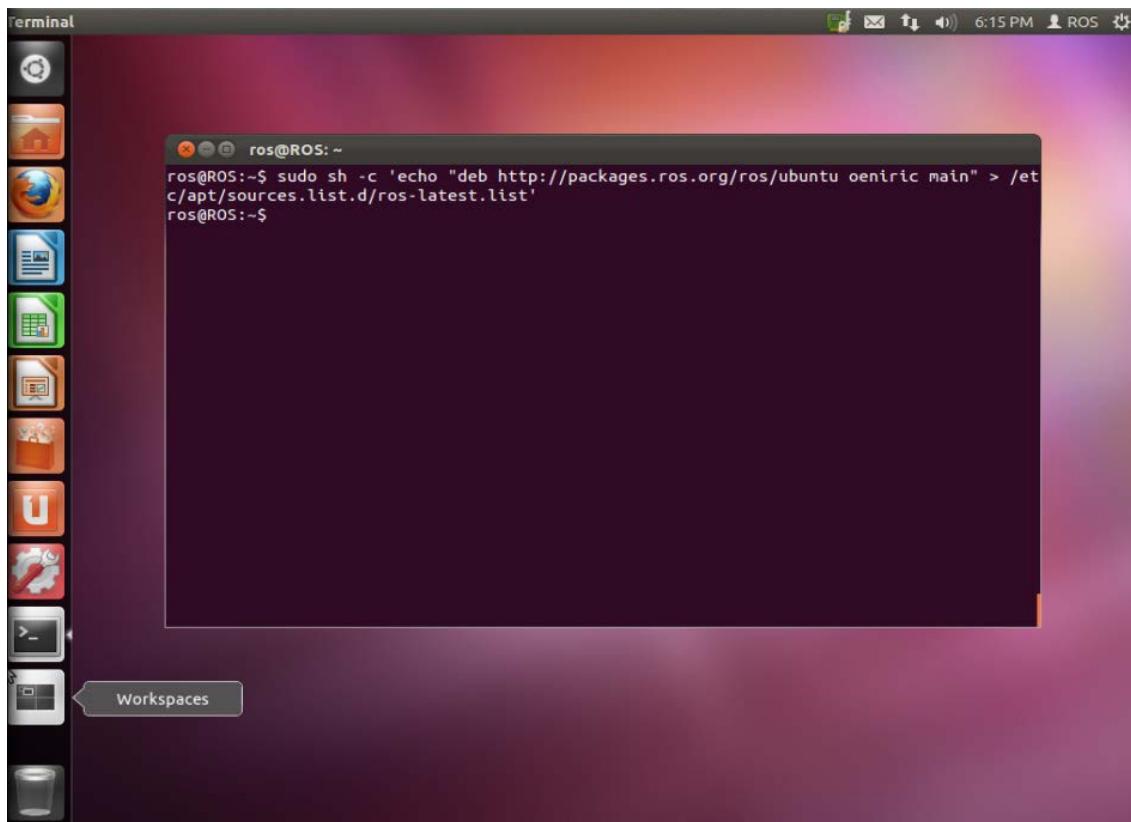
7 Instalación de ros fuerte en Ubuntu 11.10 (ONEIRIC OCELOT)

1. Ajustar los repositorios a “restricted”, “universe” y “multiverse”. Para esto se debe de ir a “Ubuntu Software Center” e instalar “Software Sources”. Sin embargo, si ud. Posee Ubuntu 9.04 (Jaunty) o superior “main”, “universe”, “restricted” y “multiverse” son habilitados por defecto. Por otro lado, si desea realizar los pasos manualmente puede revisar la siguiente dirección web:

<https://help.ubuntu.com/community.Repositories/Ubuntu>

2. Ajustar tus “sources.list”

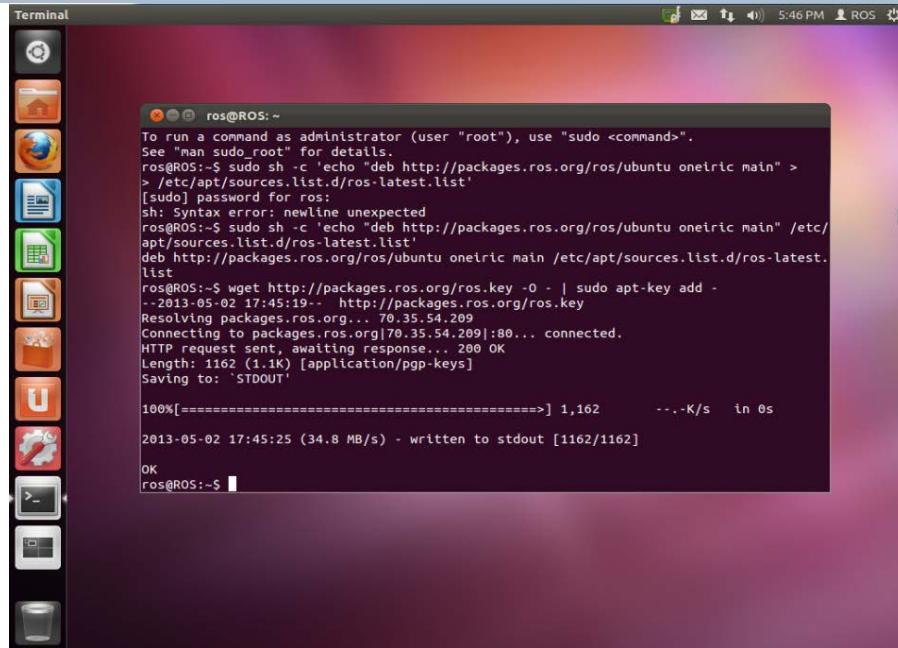
```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu oneiric main" > /etc/apt/sources.list.d/ros-latest.list'
```



¹ Actualmente la versión última de ROS (Robot Operating System) es ROS Groovy.

3. Ajustar tus llaves

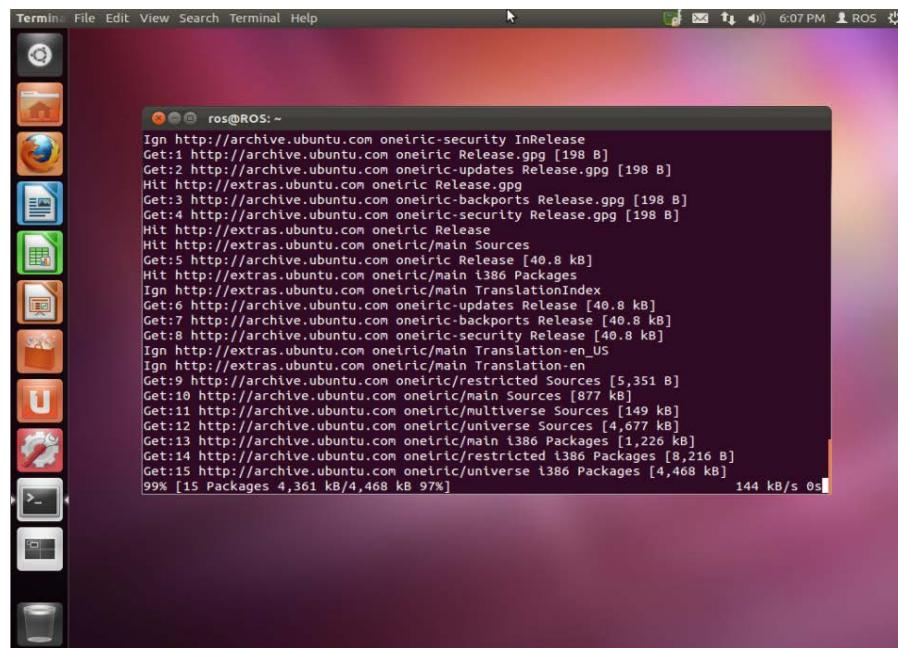
```
 wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
```



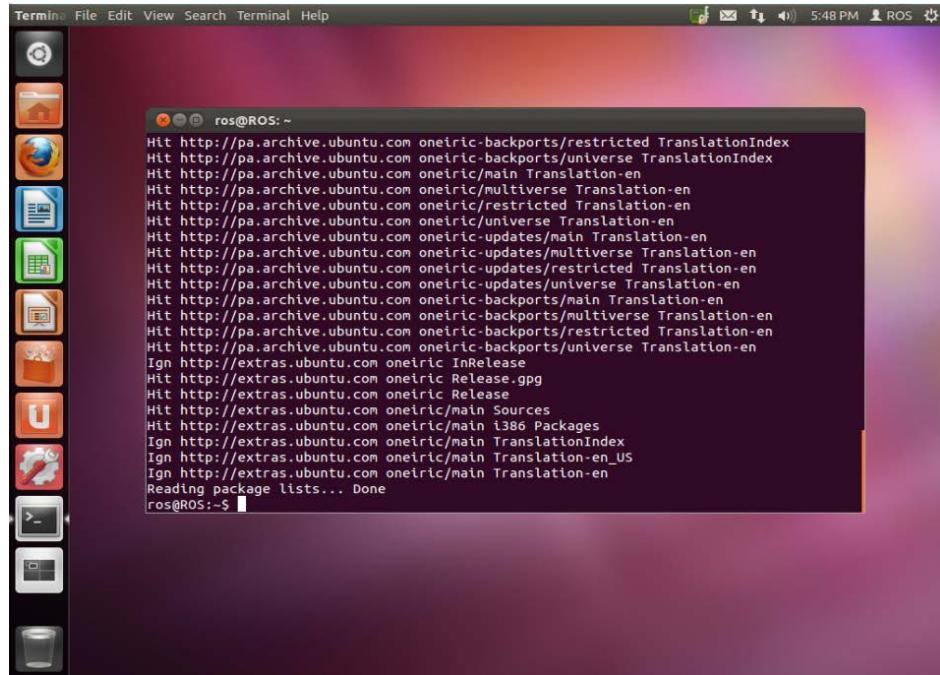
4. Instalación

Asegurarse que ud haya reindexado el servidor de ROS.org

```
sudo apt-get update
```



```
 wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
```



5. Existen muchas librerías y herramientas diferentes en ROS. Ellos proveen de forma por defecto configuraciones para iniciar a utilizarse. Ud. Puede también instalar los Stack de ROS de manera total o individualmente.

Instalación total de Escritorio: (Recomendada) ROS, rx, rvis, librerías genéricas de robot, simuladores 2D/3D, navegadores y percepción 2D/3D.

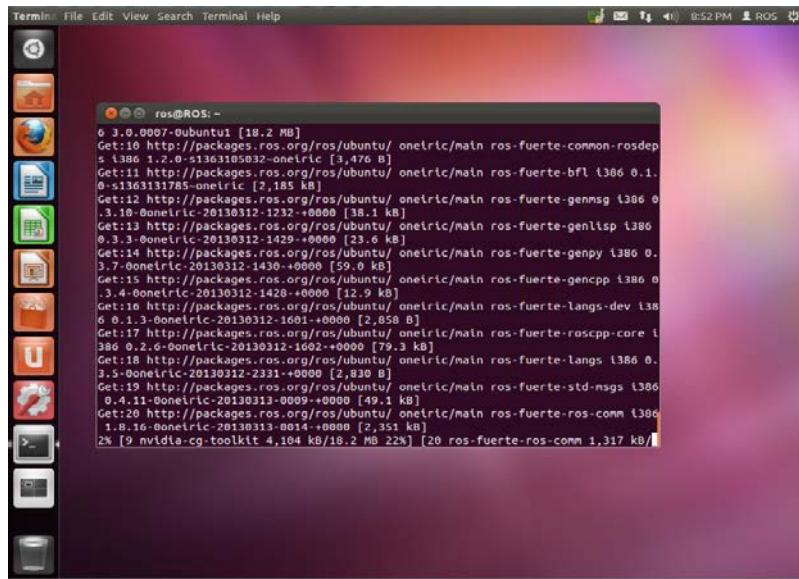
```
 sudo apt-get install ros-fuerte-desktop-full
```

Por otro lado, de manera individual se puede instalar los stacks con el siguiente comando:

```
 sudo apt-get install ros-fuerte-STACK
```

Por ejemplo, si su stack se llama “slam-gmapping”, debe poner en la terminal:

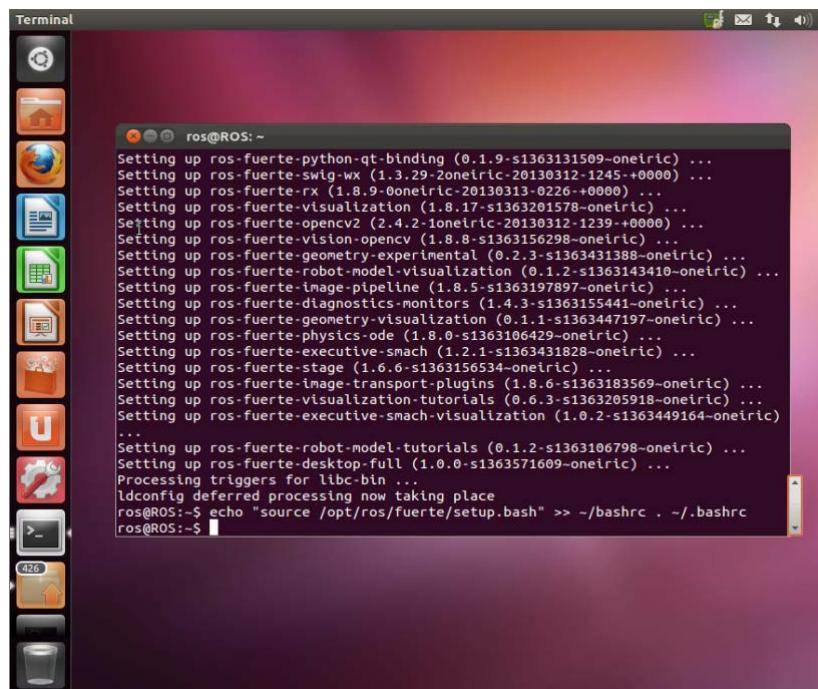
```
 sudo apt-get install ros-fuerte-slam-gmapping
```



6. Ajuste de Ambiente

Es conveniente si las variables de entorno de ROS son automáticamente añadidas a su sesión de “bash” en cada momento que un nuevo “shell” es ejecutado.

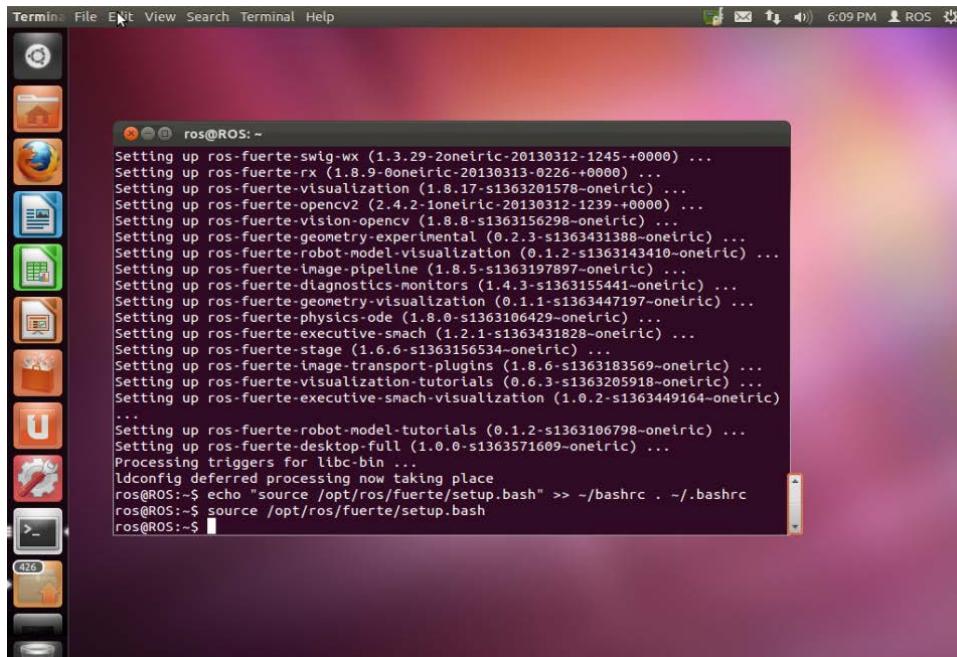
```
echo "source /opt/ros/fuerte/setup.bash" >> ~/.bashrc
. ~/.bashrc
```



Si tiene más de una distribución de ROS instalada, `~/.bashrc` DEBE SOLAMENTE apuntar al código de “`setup.bash`” de la versión que actualmente está utilizando.

Si solamente quiere cambiar al ambiente de desarrollo de ROS, puede insertar desde el teclado:

```
source /opt/ros/fuerte/setup.bash
```



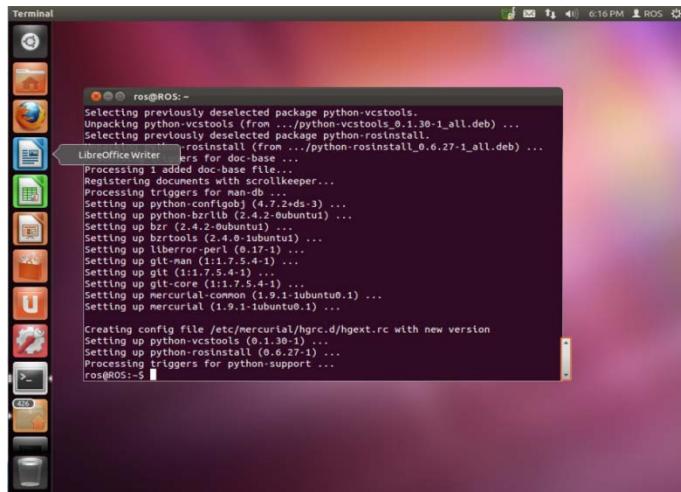
La mayoría de los usuarios siempre quieren crear sus propios paquetes. La mejor manera de realizarlo es configurando el entorno de desarrollo y crear un “overlay”.

7. Herramientas independientes (Standalone)

rosinstall y rosdep son frecuentemente utilizadas como herramientas de línea de comandos en ROS y son distribuidas por separado. Rosinstall habilita a ud. bajar fácilmente árboles de códigos fuentes de pilas (stacks) de ROS y paquetes. rosdep le hace fácil la instalación de dependencias del sistema del código fuente que está por compilar.

Para instalar estas herramientas, correr:

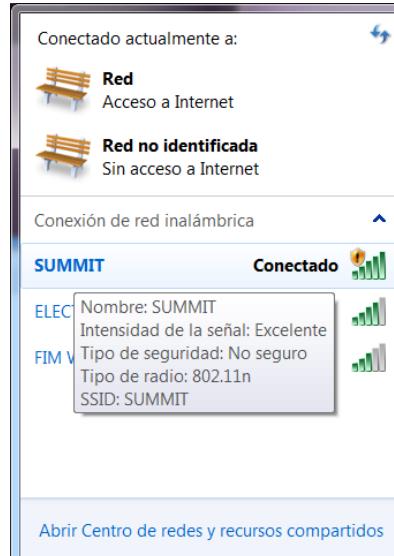
```
sudo apt-get install python-rosinstall python-rosdep
```

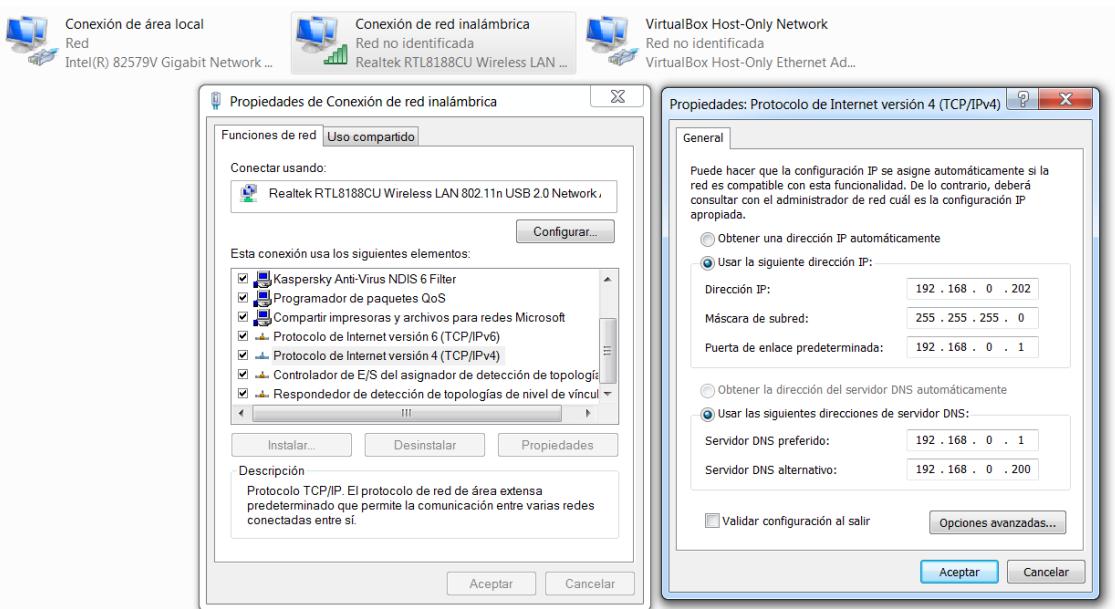


8. Probando conectividad.

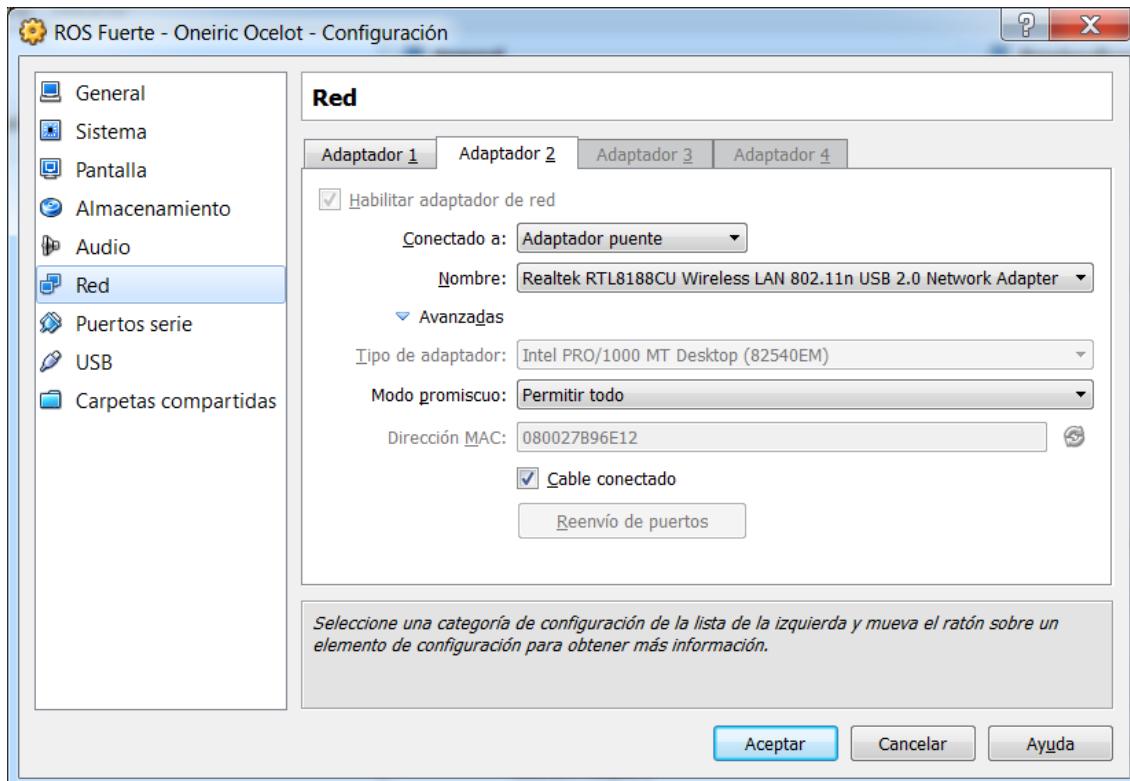
Para enlazar los equipos debe poner tanto al Summit XL como su PC en la misma red.

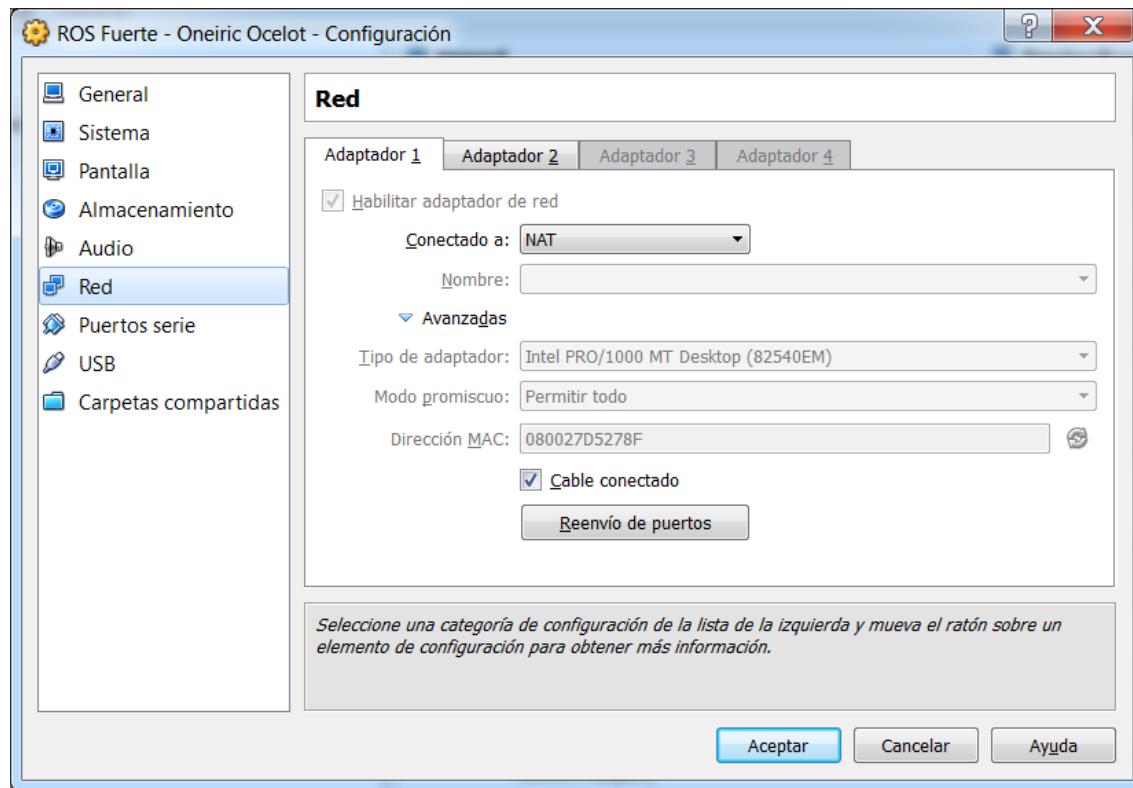
Primeramente ajustamos a la PC dentro de la misma red del Summit XL por medio del adaptador inalámbrico. Solamente son necesarios los parámetros de IP, máscara de red. Los demás parámetros son innecesarios.





Bajo nuestro procedimiento estamos utilizando una máquina virtual. Nuestro requerimiento fue que todo quedara bajo una misma red y también poseer internet. En otras palabras, tanto la PC con sistema operativo Windows, como la máquina virtual con ROS Fuerte y el robot Summit XL deben quedar en la misma subred; y no debo perder conectividad a internet. Para ello se debe ajustar el adaptador de red de la siguiente manera:





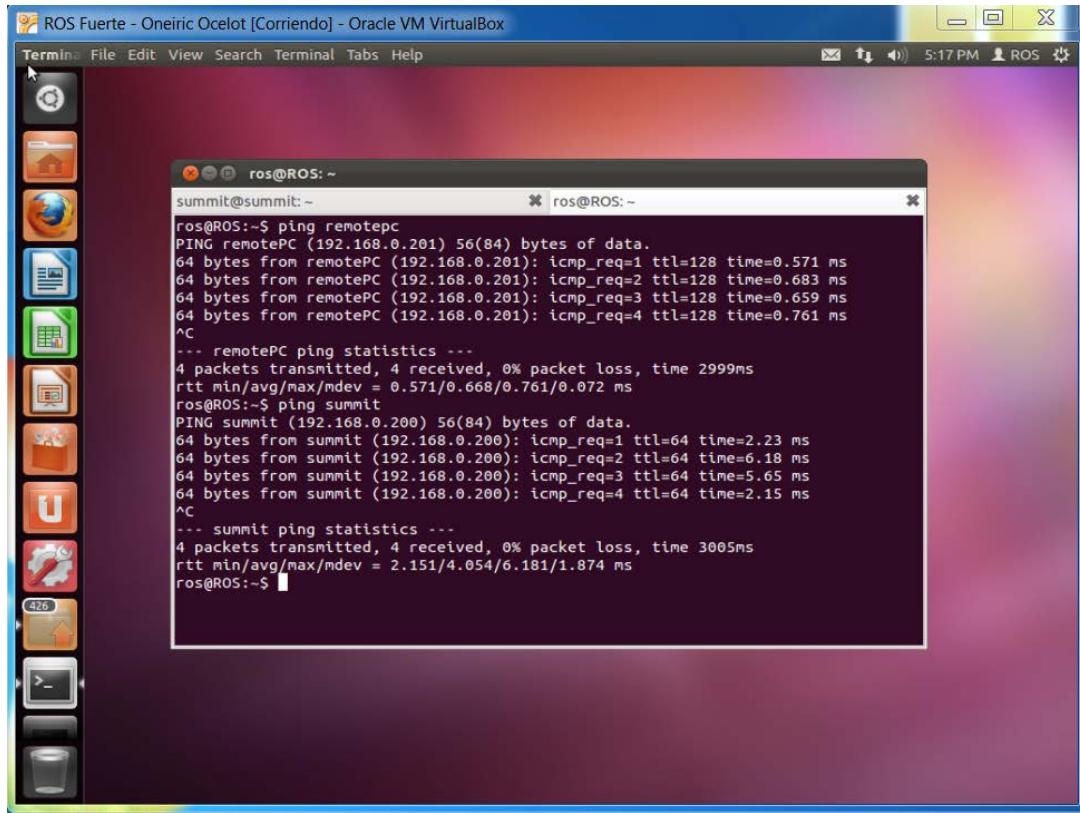
Primeramente agregue los dispositivos en la sección de hosts en su computador con Linux el cual llamaremos “remotePC” y al Summit XL lo denominaremos “summit”.

```
ros@ROS:~$ sudo gedit /etc/hosts
hosts (/etc) - gedit
File Edit View Search Tools Documents Help
Office Writer Open Save Undo Vw Folders Search
hosts x
127.0.0.1      localhost
127.0.1.1      ROS
192.168.0.200   summit
192.168.0.201   remotePC

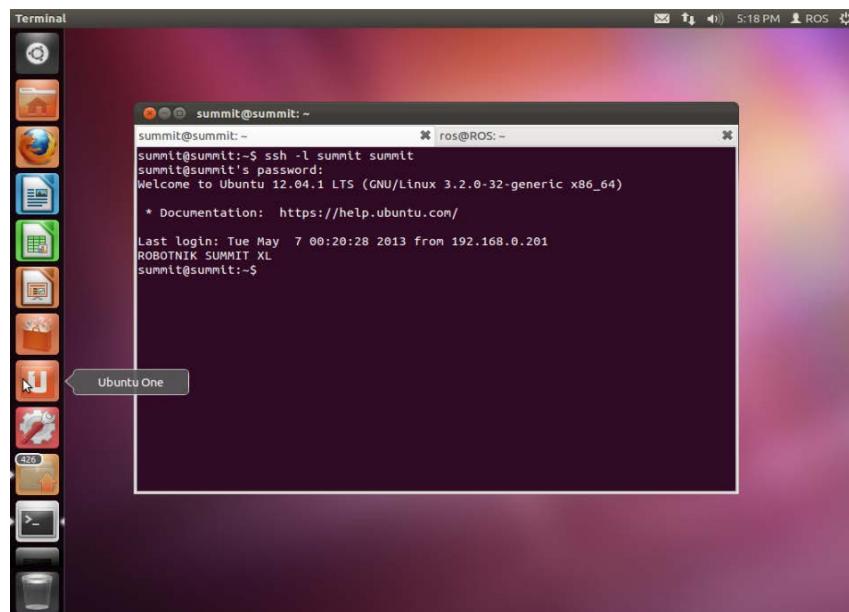
# The following lines are desirable for IPv6 capable hosts
::1      ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

Plain Text ▾ Tab Width: 8 ▾ Ln 1, Col 1 INS
```

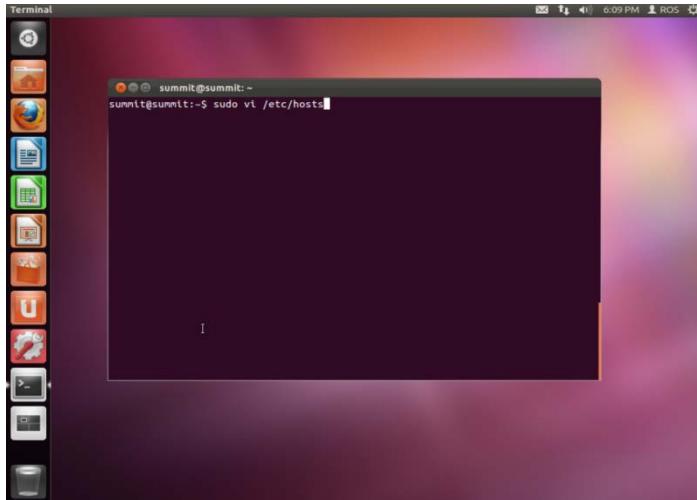
Seguido pruebe conectividad entre ambos equipos. Hacer ping desde la PC en Linux a "Summit" y a nuestra PC "remotePC". Si todo está bien coordinado, ambas deben verse en la misma red sin problemas según muestra la figura.



Seguido repetiremos el mismo procedimiento mediante una conexión segura, para ello, ingrese al Summit XL de la siguiente manera:



Cambiaremos los parámetros de host en dicho dispositivo utilizando el visor vim (vi), el password para el robot Summit XL es *summit*:



Al abrirse el editor de vim, necesitamos insertar igualmente los parámetros de red de los dispositivos. Para añadir dispositivos utilizando este visor de Linux, recomendamos leer un tutorial de vi antes.

```
summit@summit:~$ cat /etc/hosts
127.0.0.1      localhost
127.0.1.1      summit
192.168.0.200  summit
192.168.0.3    trurl
192.168.0.201  remotePC

# The following lines are desirable for IPv6 capable hosts
::1      ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

"/etc/hosts" 12L, 293C          5,1          All
```

9. Creando el Espacio de Trabajo

Antes de crear nuestro espacio de trabajo debemos instalar y ajustar los archivos *sh en '/opt/ros/fuerte/' , y los podemos adjuntar con el siguiente comando:

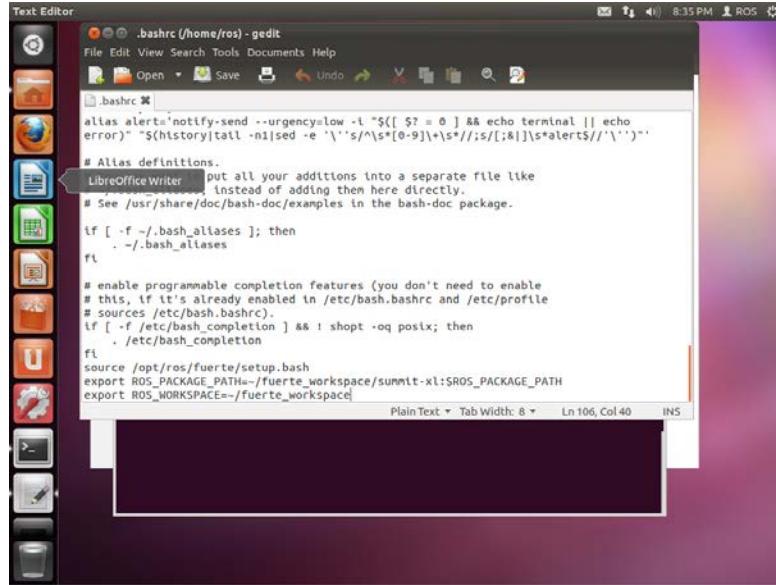
```
source /opt/ros/fuerte/setup.bash
```

Cada vez que deseemos usar ROS en una nueva terminal, debemos realizar la inicialización de acción al bash. Este proceso permite instalar muchas distribuciones de ROS y conmutar entre ellas.

Cuando queremos desarrollar con código de ROS, es usual crear un espacio de trabajo. Para esto utilizaremos rosdep que es una herramienta que provee una interfaz uniforme entre varios

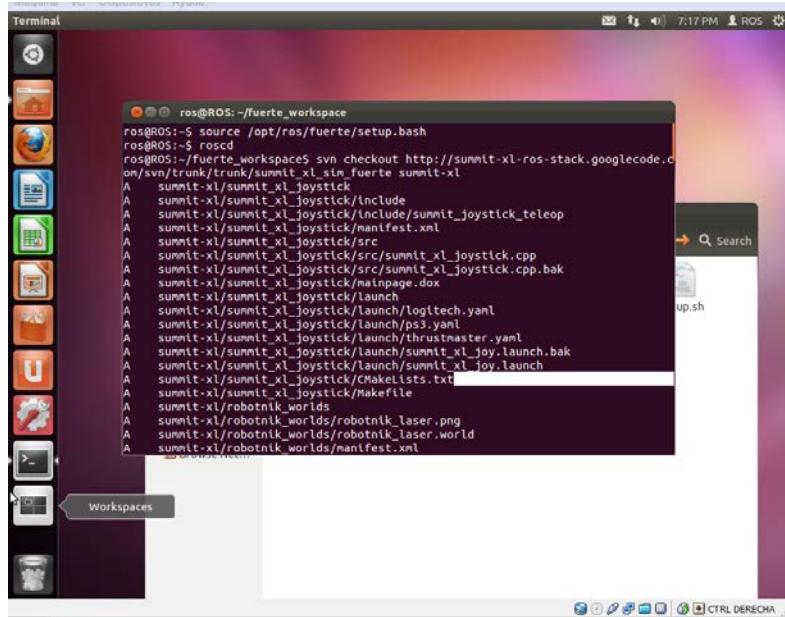
sistemas de control de versión como SVN, Git y Mercurial administrando los paquetes de ros en sobrecapas. Inicialice el espacio de trabajo y ruta de paquetes bajo los siguientes comandos:

```
rosws init ~/fuerte_workspace /opt/ros/fuerte
echo "source /opt/ros/fuerte/setup.bash" >> ~/.bashrc
echo "export ROS_PACKAGE_PATH=~/fuerte_workspace/summit-xl:$ROS_PACKAGE_PATH" >> ~/.bashrc
echo "export ROS_WORKSPACE=~/fuerte_workspace" >> ~/.bashrc
```

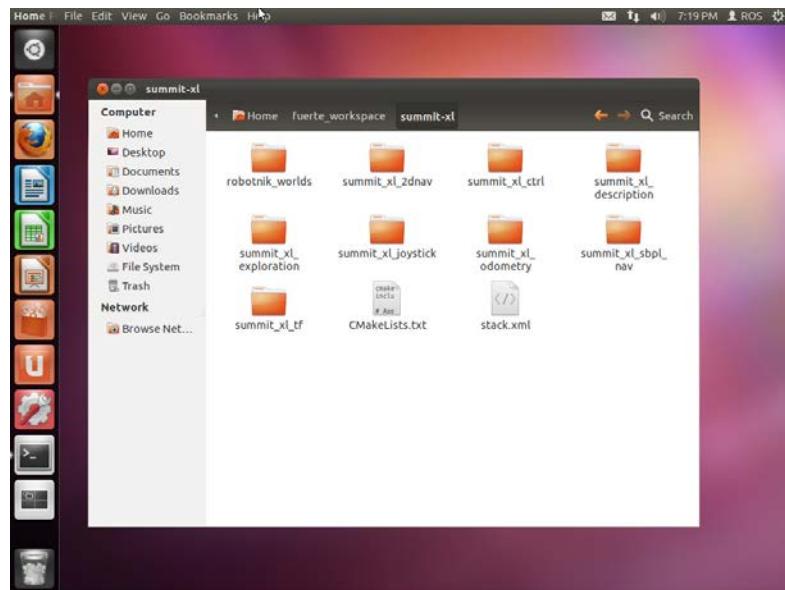


Ahora instalaremos los paquetes de software de simulación del repositorio. Para ello ejecutaremos la siguiente línea de comandos:

```
svn checkout http://summit-xl-ros-stack.googlecode.com/svn/trunk/trunk/summit_xl_sim_fuerte summit-xl
```

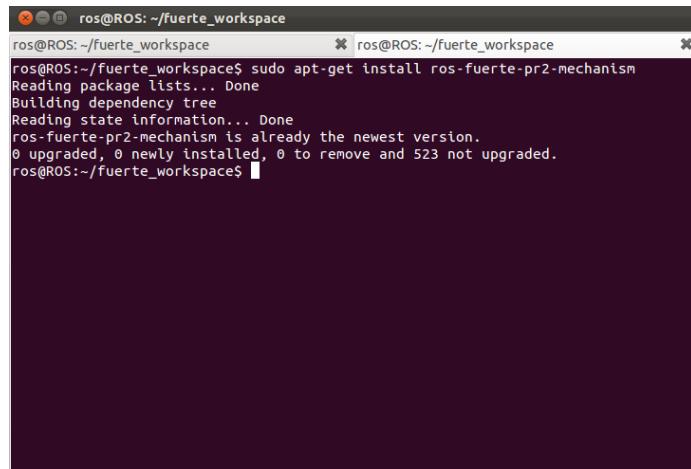


Buscar en la carpeta de su espacio de trabajo y verificar que los archivos hayan sido descargados:



Antes de ejecutar “cmake .” dentro de cada carpeta, debemos bajar el paquete de software “pr2_mechanism” que son dependencias de este robot. Para esto realizamos la siguiente acción que instala esta stack:

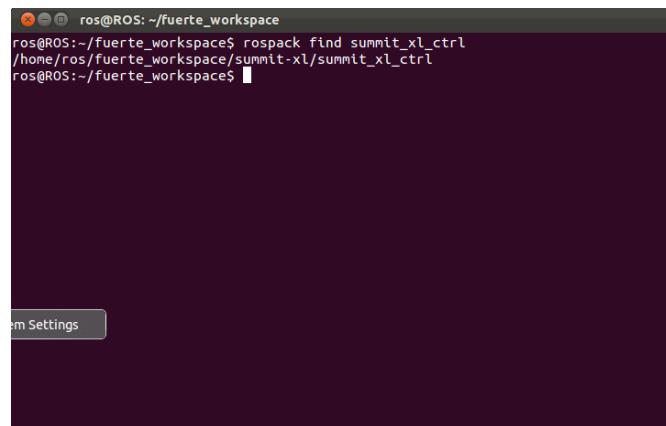
```
sudo apt-get install ros-fuerte-pr2-mechanism
```



```
ros@ROS:~/fuerte_workspace$ sudo apt-get install ros-fuerte-pr2-mechanism
Reading package lists... Done
Building dependency tree
Reading state information... Done
ros-fuerte-pr2-mechanism is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 523 not upgraded.
ros@ROS:~/fuerte_workspace$
```

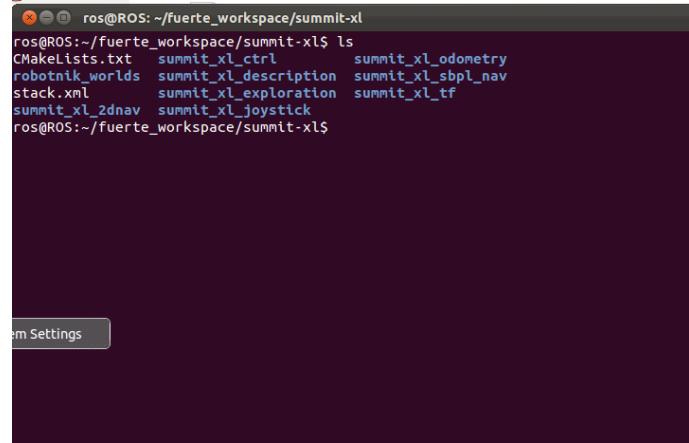
Verificar que ROS encuentra los paquetes de simulación del robot

```
rospack find summit_xl_ctrl
```

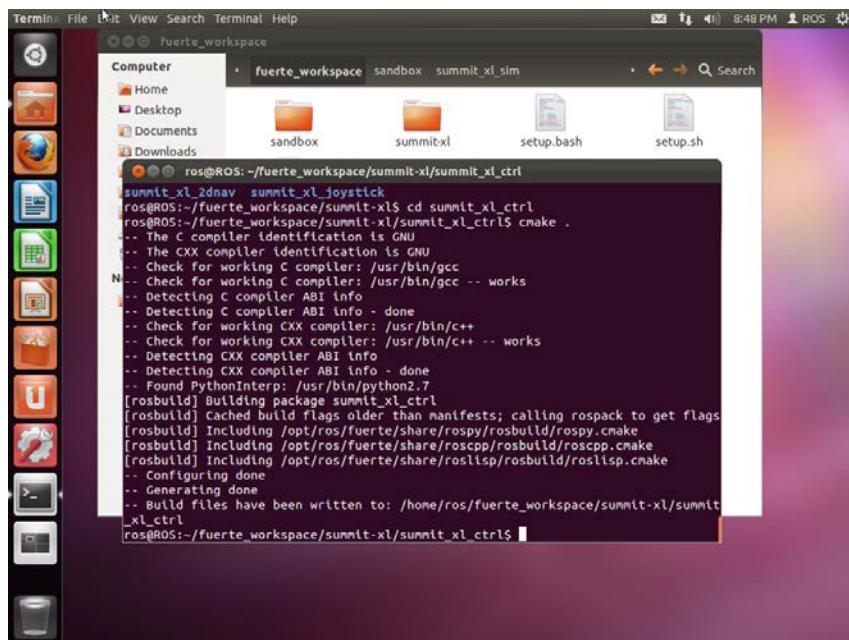


```
ros@ROS:~/fuerte_workspace$ rospack find summit_xl_ctrl
/home/ros/fuerte_workspace/summit-xl/summit_xl_ctrl
ros@ROS:~/fuerte_workspace$
```

Una vez hecho esto, ejecutar el comando "cmake ." en todas las carpetas para que se vuelvan a generar los Makefile de cada paquete con prefijo summit_xl que se encuentran dentro de la carpeta summit-xl:



```
ros@ROS: ~/fuerte_workspace/summit-xl
ros@ROS:~/fuerte_workspace/summit-xl$ ls
CMakeLists.txt    summit_xl_ctrl      summit_xl_odometry
robotnik_worlds  summit_xl_description  summit_xl_sbpl_nav
stack.xml        summit_xl_exploration  summit_xl_tf
summit_xl_2dnav  summit_xl_joystick
```

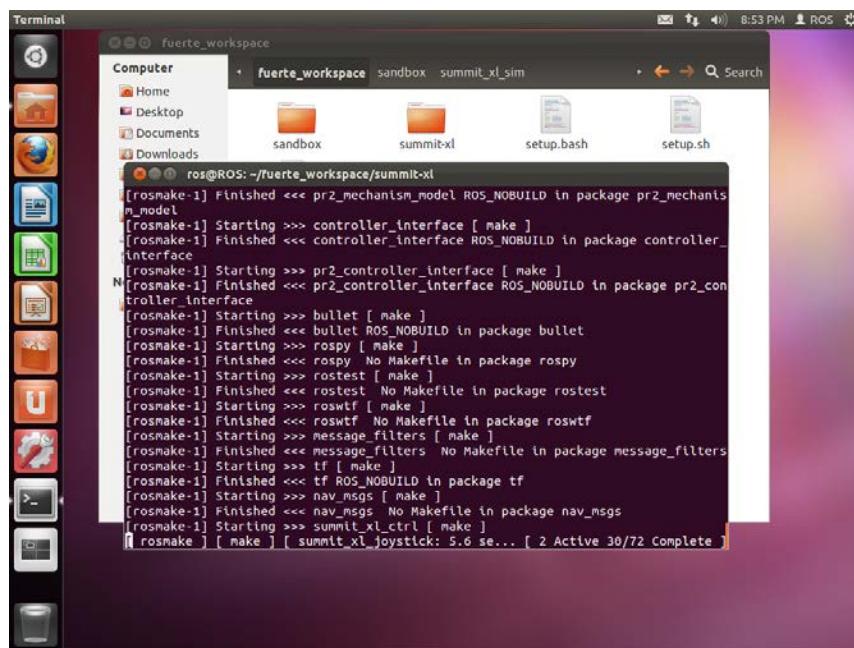
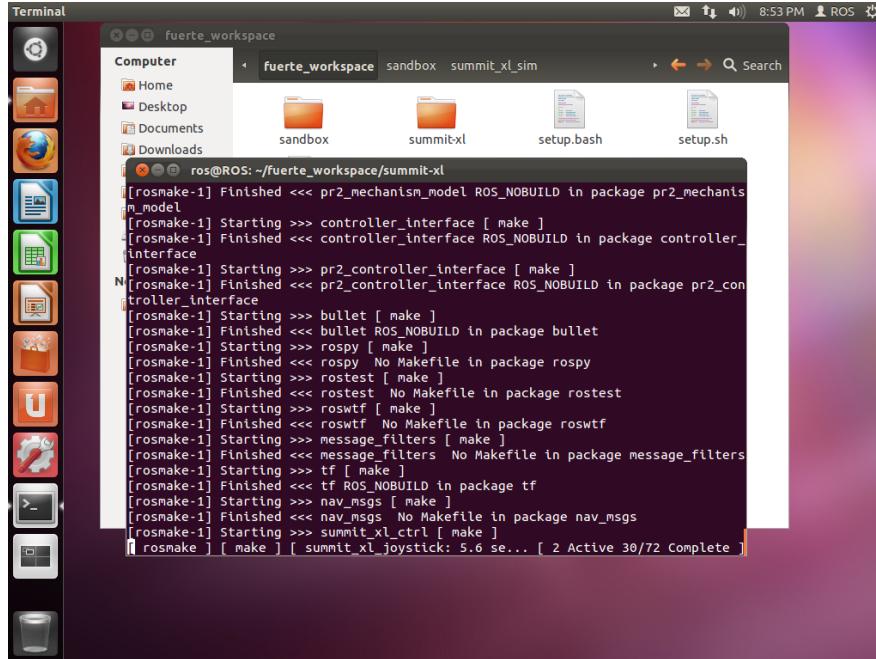


```
Terminal File View Search Terminal Help
File fuerte_workspace
Computer
  Home
  Desktop
  Documents
  Downloads
  sandbox
  summit_xl
  setup.bash
  setup.sh
Search
8:48 PM ROS

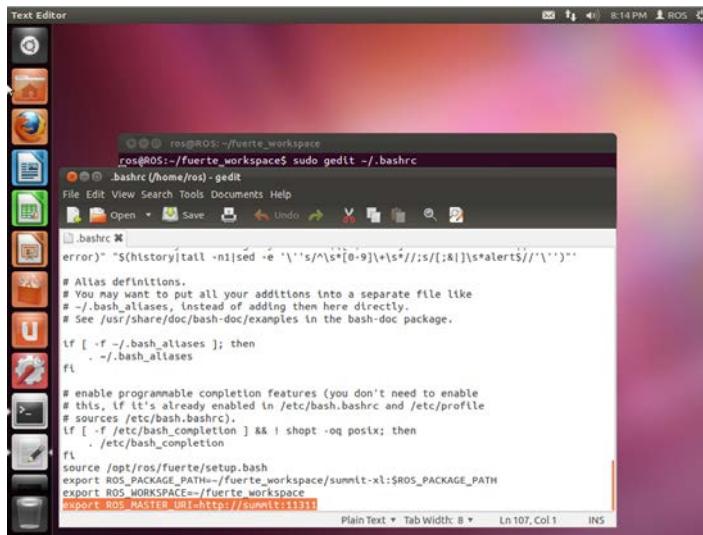
ros@ROS: ~/fuerte_workspace/summit_xl/summit_xl_ctrl
summit_xl_2dnav  summit_xl_joystick
ros@ROS:~/fuerte_workspace/summit_xl$ cd summit_xl_ctrl
ros@ROS:~/fuerte_workspace/summit_xl$ cmake .
-- The C compiler identification is GNU
-- The CXX compiler identification is GNU
-- Check for working C compiler: /usr/bin/gcc -- works
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ --
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Found PythonInterp: /usr/bin/python2.7
[rosbuild] Building package summit_xl_ctrl
[rosbuild] Cached build flags older than manifests; calling rospack to get flags
[rosbuild] Including /opt/ros/fuerte/share/rospy/rospack/rospy.cmake
[rosbuild] Including /opt/ros/fuerte/share/roscpp/rospack/roscpp.cmake
[rosbuild] Including /opt/ros/fuerte/share/roslisp/rospack/roslisp.cmake
-- Configuring done
-- Generating done
-- Build files have been written to: /home/ros/fuerte_workspace/summit_xl/summit_xl_ctrl
ros@ROS:~/fuerte_workspace/summit_xl$
```

Ejecutar "cmake ." también en la carpeta **summit_xl_exploration/explore** .

Por último, en la raíz del directorio summit-xl, ejecutar "rosmake --pre-clean", para generar todos los ejecutables y poder comenzar a utilizar la simulación.



Seguido, para realizar acciones remotas desde una PC con el robot corriendo ROS, se debe modificar el archivo de "bash"



Bajo esta modificación se pueden llamar acciones remotas desde la “remotePC” hacia el Summit XL.

Finalmente para que tome las modificaciones, en la terminal ejecutar la siguiente sentencia:

```
. ~/.bashrc
```

Para ver los tópicos de la IMU (Inertial Measurement Unit)

```
rostopic echo /imu/data
```

```
ros@ROS: ~/fuerte_workspace
header:
  seq: 178382
  stamp:
    secs: 1370395830
    nsecs: 717091568
  frame_id: imu
orientation:
  x: -0.0158958298608
  y: -0.0160403800691
  z: -0.789762105344
  w: 0.610236531905
orientation_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
angular_velocity:
  x: -0.0021956909565
  y: 0.0028818443804
  z: 0.00233292164128
angular_velocity_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
linear_acceleration:
  x: 0.506529390812
  y: 0.0577058829367
  z: 10.3678236008
linear_acceleration_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

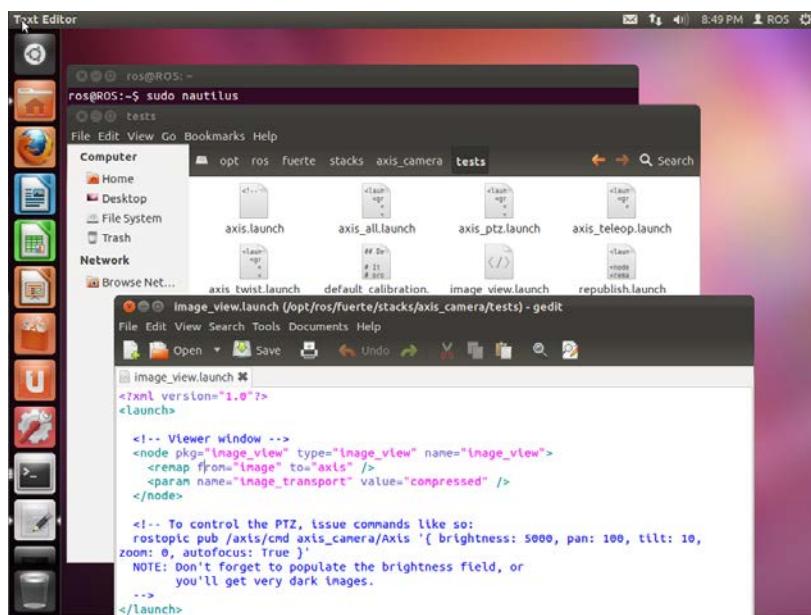
Si quiere probar si la cámara está captando información en tiempo real, podemos realizar lo siguiente:

Bajar el paquete de ros axis_camera del repositorio a las carpetas de ROS

```
sudo apt-get install ros-fuerte-axis-camera
```



Navegamos hasta la carpeta local de nuestra computadora axis_camera con roscd y creamos un archivo “*launch*” de configuración de la cámara para poder conectarnos de manera remota:



Finalizado esto nos conectamos bajo una terminal de manera remota con el robot por ssh y lanzamos el controlador desde el summit:

```

ssh -l summit summit
summit@summit's password: summit
rosrun axis_camera axis_ptz.launch

```

```

summit@summit:~  

ros@ROS:-$ ssh -l summit summit  

summit's password:  

Welcome to Ubuntu 12.04.1 LTS (GNU/Linux 3.2.0-32-generic x86_64)  

 * Documentation: https://help.ubuntu.com/  

Last login: Wed Jun 5 04:00:28 2013 from remotepc  

ROBOTNIK SUMMIT XL  

summit@summit:~$ rosrun axis_camera axis_ptz.launch

```

```

/opt/ros/fuerte/stacks/summit_xl_robot/axis_camera/axis_ptz.launch http://localhost:11311
* /axis/width
* /rosdistro
* /rosversion
/axis/
  axis (axis_camera/axis.py)
  axis_ptz (axis_camera/axis_ptz.py)

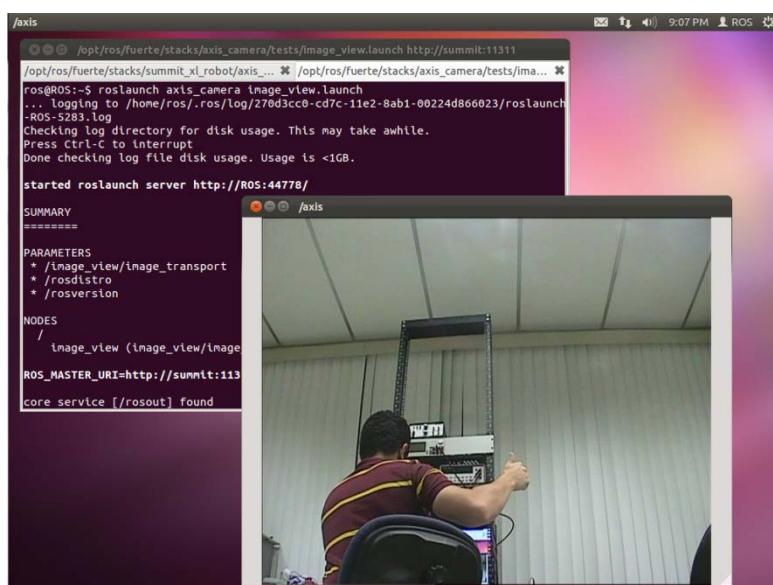
ROS_MASTER_URI=http://localhost:11311

core service [/rosout] found
Exception AttributeError: 'AttributeError("'.DummyThread' object has no attribute '_Thread_block'")' in <module 'threading' from '/usr/lib/python2.7/threading.py'>: ignored
process[axis-1]: started with pid [7245]
Exception AttributeError: 'AttributeError("'.DummyThread' object has no attribute '_Thread_block'")' in <module 'threading' from '/usr/lib/python2.7/threading.py'>: ignored
process[axis/axis_ptz-2]: started with pid [7246]
AxisPTZ: hostname=192.168.0.185
AxisPTZ: username=root
AxisPTZ: password=summitxl

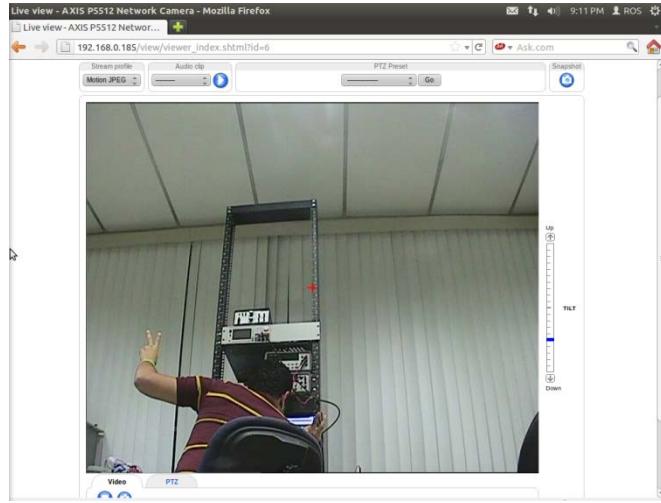
```

En otra terminal (Shift+Ctrl+T), es decir, desde la PC remota o local, lanzamos y verificamos la imagen de la cámara con rosrun:

```
rosrun axis_camera image_view.launch
```

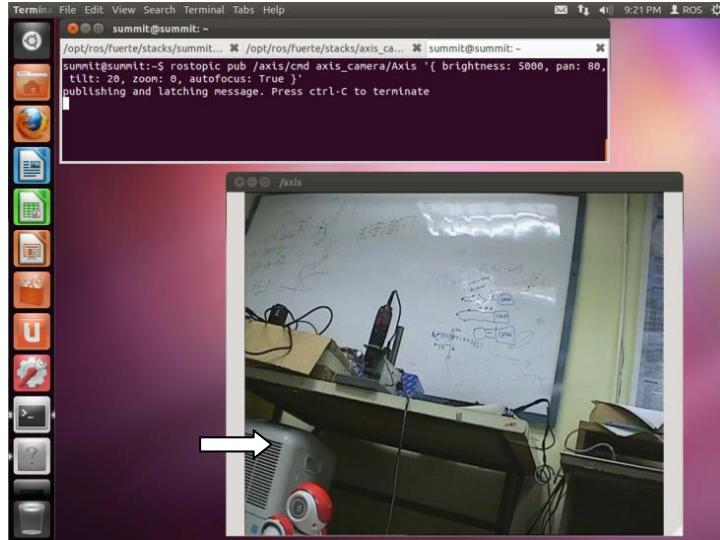


También puede verificar que la terminal web está andando insertando en el navegador la siguiente dirección IP: 192.168.0.185.



Finalmente, para concluir la prueba de la cámara, pruebe con rostopic cambiar ciertos ajustes, para esto, en otra terminal ingresamos al robot y publicamos el siguiente comando:

```
rostopic pub /axis/cmd axis_camera/Axis '{ brightness: 5000, pan: 80, tilt: 20, zoom: 0, autofocus: True }'
```



Para ejecutar la acción del robot con el control de PS3, se debe primero

- Copiar las carpetas `summit_xl_ctrl` y `summit_xl_sim` del robot a la pc remota
- Copiar la carpeta `summit_xl_pad` dentro de la carpeta de `summit-xl`
- Compilar con cmake:
 - `cd summit_xl_pad`

- rm CMakeLists.txt
- cmake .
- rosmake

Para ejecutar un mando de movimiento sobre el controlador del robot podemos publicar el siguiente mensaje en el tópico. De esta manera ejecutaremos una opción que realice una trayectoria lineal hacia adelante:

```
rostopic pub -r 10 summit_xl_controller/command geometry_msgs/Twist '{linear: {x: 0.6, y: 0, z: 0}, angular: {x: 0, y: 0, z: 0}}'
```

Para ejecutar un mando de movimiento sobre el controlador del robot podemos publicar el siguiente mensaje en el tópico. De esta manera ejecutaremos una opción que realice una trayectoria lineal hacia atrás:

```
rostopic pub -r 10 summit_xl_controller/command geometry_msgs/Twist '{linear: {x: -0.6, y: 0, z: 0}, angular: {x: 0, y: 0, z: 0}}'
```

Para ejecutar un mando de movimiento sobre el controlador del robot podemos publicar el siguiente mensaje en el tópico. De esta manera ejecutaremos una opción que realice una trayectoria angular hacia adelante:

```
rostopic pub -r 10 summit_xl_controller/command geometry_msgs/Twist '{linear: {x: 0, y: 0, z: 0}, angular: {x: 0, y: 0, z: 0.6}}'
```

Para ejecutar un mando de movimiento sobre el controlador del robot podemos publicar el siguiente mensaje en el tópico. De esta manera ejecutaremos una opción que realice una trayectoria angular hacia atrás:

```
rostopic pub -r 10 summit_xl_controller/command geometry_msgs/Twist '{linear: {x: 0, y: 0, z: 0}, angular: {x: 0, y: 0, z: -0.6}}'
```

8 Controlador de Velocidad del Summit XL (Summit_xl_controller.h)

```
/*
 * summit_xl_ctrl
 * Copyright (c) 2012, Robotnik Automation, SLL
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * * Redistributions of source code must retain the above copyright
 *   notice, this list of conditions and the following disclaimer.
 * * Redistributions in binary form must reproduce the above copyright
 *   notice, this list of conditions and the following disclaimer in the
 *   documentation and/or other materials provided with the distribution.
 * * Neither the name of the Robotnik Automation, SLL. nor the names of its
```

```

*   contributors may be used to endorse or promote products derived from
*   this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
* POSSIBILITY OF SUCH DAMAGE.
*
* \author Marc Benet3 (mbeneto@robotnik.es)
* \brief Relate the 4-wheels joints with each motor, apply the control correction in closed loop for
the motor and set the command received by the
joystick/keyboard teleop node.
*/

```

```

#include <pr2_controller_interface/controller.h>
#include <pr2_mechanism_model/joint.h>
#include <geometry_msgs/Twist.h>
#include <tf/transform_broadcaster.h>
#include <nav_msgs/Odometry.h>

#define SUMMIT_XL_D_TRACKS_M 1.0 // check !
#define SUMMIT_XL_WHEEL_DIAMETER 0.25

namespace summit_xl_ctrl_ns
{
/*!\class
\brief This class inherits from Controller and implements the actual controls.
*/
class SummitXLControllerClass: public pr2_controller_interface::Controller
{
private:
// Joint states
pr2_mechanism_model::JointState* joint_state_blw_;
pr2_mechanism_model::JointState* joint_state_flw_;
pr2_mechanism_model::JointState* joint_state_brw_;
pr2_mechanism_model::JointState* joint_state_frw_;
// Joint Positions
double init_pos_blw_;
double init_pos_flw_;
double init_pos_brw_;

```

```

double init_pos_frw_;
// Joint Speeds
double init_vel_blw_;
double init_vel_flw_;
double init_vel_brw_;
double init_vel_frw_;

// Robot Speeds
double linearSpeedMps_;
double angularSpeedRads_;

// Robot Positions
double robot_pose_px_;
double robot_pose_py_;
double robot_pose_pa_;

// External speed references
double v_ref_;
double w_ref_;

// Left-right reference
int left_right;

// Frequency
double frequency;

// Speed references for motor control
double v_left_mps, v_right_mps;

<太后!
 * \brief callback message, used to remember where the base is commanded to go
 */
geometry_msgs::Twist base_vel_msg_;

```

```

//deal with Twist commands
void commandCallback (const geometry_msgs::TwistConstPtr &msg);

private:
    double saturation(double u, double min, double max);

};

}

```

9 Controlador de Velocidad del Summit XL (summit_xl_controller.cpp)

```

/*
 * summit_xl_ctrl
 * Copyright (c) 2012, Robotnik Automation, SLL
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 *   * Redistributions of source code must retain the above copyright
 *     notice, this list of conditions and the following disclaimer.
 *   * Redistributions in binary form must reproduce the above copyright
 *     notice, this list of conditions and the following disclaimer in the
 *     documentation and/or other materials provided with the distribution.
 *   * Neither the name of the Robotnik Automation, SLL. nor the names of its
 *     contributors may be used to endorse or promote products derived from
 *     this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 *
 * \author Marc Benetó (mbeneto@robotnik.es)
 * \brief Relate the 4-wheels joints with each motor, apply the control correction in closed loop for
 * the motor and set the command received by the joystick/keyboard teleop node.
 */

```

```

#include "summit_xl_ctrl/summit_xl_ctrl.h"
#include <pluginlib/class_list_macros.h>

```

```

namespace summit_xl_ctrl_ns {

bool SummitXLControllerClass::init(pr2_mechanism_model::RobotState *robot,
                                   ros::NodeHandle &n)
{
// 4-Axis Skid Steer Rover

// back left wheel
ROS_INFO("summit_xl_ctrl - starting controller");
std::string joint_blw;
if (!n.getParam("joint_blw", joint_blw))
{
    ROS_ERROR("No joint BLW given in namespace: '%s'",
              n.getNamespace().c_str());
    return false;
}
joint_state_blw_ = robot->getJointState(joint_blw);
if (!joint_state_blw_)
{
    ROS_ERROR("summit_xl_ctrl could not find joint named '%s'",
              joint_blw.c_str());
    return false;
}
ROS_DEBUG("summit_xl_ctrl found joint named '%s'", joint_blw.c_str());

// front left wheel
std::string joint_flw;
if (!n.getParam("joint_flw", joint_flw))
{
    ROS_ERROR("No joint FLW given in namespace: '%s'",
              n.getNamespace().c_str());
    return false;
}
joint_state_flw_ = robot->getJointState(joint_flw);
if (!joint_state_flw_)
{
    ROS_ERROR("summit_xl_ctrl could not find joint named '%s'",
              joint_flw.c_str());
    return false;
}
ROS_DEBUG("summit_xl_ctrl found joint named '%s'", joint_flw.c_str());

// back right wheel
std::string joint_brw;
if (!n.getParam("joint_brw", joint_brw))

```

```

{
    ROS_ERROR("No joint BRW given in namespace: '%s')",
        n.getNamespace().c_str());
    return false;
}
joint_state_brw_ = robot->getJointState(joint_brw);
if (!joint_state_brw_)
{
    ROS_ERROR("summit_xl_ctrl could not find joint named '%s'",
        joint_brw.c_str());
    return false;
}
ROS_DEBUG("summit_xl_ctrl found joint named '%s'", joint_brw.c_str());

// front right wheel
std::string joint_frw;
if (!n.getParam("joint_frw", joint_frw))
{
    ROS_ERROR("No joint FRW given in namespace: '%s')",
        n.getNamespace().c_str());
    return false;
}
joint_state_frw_ = robot->getJointState(joint_frw);
if (!joint_state_frw_)
{
    ROS_ERROR("summit_xl_ctrl could not find joint named '%s'",
        joint_frw.c_str());
    return false;
}
//ROS_ERROR("summit_xl_ctrl found joint named '%s'", joint_frw.c_str());

// Robot Speeds
linearSpeedMps_ = 0.0;
angularSpeedRads_ = 0.0;

// Robot Positions
robot_pose_px_ = 0.0;
robot_pose_py_ = 0.0;
robot_pose_pa_ = 0.0;

// External speed references
v_ref_ = 0.0;
w_ref_ = 0.0;

// Frequency
frequency = 100.0; // Hz 50.0

// for later access

```

```

node_ = n;

cmd_sub_ = node_.subscribe<geometry_msgs::Twist>("command", 1,
&SummitXLControllerClass::commandCallback, this);

return true;
}

/// Controller startup in realtime
void SummitXLControllerClass::starting()
{
// Initial positions
init_pos_blw_ = joint_state_blw_->position_;
init_pos_flw_ = joint_state_flw_->position_;
init_pos_brw_ = joint_state_brw_->position_;
init_pos_frw_ = joint_state_frw_->position_;
// Initial velocities
init_vel_blw_ = joint_state_blw_->velocity_;
init_vel_flw_ = joint_state_flw_->velocity_;
init_vel_brw_ = joint_state_brw_->velocity_;
init_vel_frw_ = joint_state_frw_->velocity_;

}

/// Controller update loop in realtime
void SummitXLControllerClass::update()
{
double v_left_mps, v_right_mps;

// Calculate its own velocities for realize the motor control
v_left_mps = ((joint_state_blw_->velocity_ + joint_state_flw_->velocity_) / 2.0) *
(SUMMIT_XL_WHEEL_DIAMETER / 2.0);
v_right_mps = -((joint_state_brw_->velocity_ + joint_state_frw_->velocity_) / 2.0) *
(SUMMIT_XL_WHEEL_DIAMETER / 2.0); // sign according to urdf (if wheel model is not symetric,
should be inverted)

linearSpeedMps_ = (v_right_mps + v_left_mps) / 2.0; // m/s
angularSpeedRads_ = (v_right_mps - v_left_mps) / SUMMIT_XL_D_TRACKS_M; // rad/s

//ROS_INFO("linearSpeedMps=%5.2f, angularSpeedRads=%5.2f", linearSpeedMps_,
angularSpeedRads_);

// Current AX3500 controllers close this loop allowing (v,w) references.
double epv=0.0;
double epw=0.0;

```

```

static double epvant =0.0;
static double epwant =0.0;

// Adjusted for soft indoor office soil
double kpv=10.0; double kdv=0.0;
double kpw=20.0; double kdw=0.0;

// State feedback error
epv = v_ref_ - linearSpeedMps_;
epw = w_ref_ - angularSpeedRads_;

// Compute state control actions
double uv= kpv * epv + kdv * (epv - epvant);
double uw= kpw * epw + kdw * (epw - epwant);
epvant = epv;
epwant = epw;

// Inverse kinematics
double dUl = uv - 0.5 * SUMMIT_XL_D_TRACKS_M * uw;
double dUr = -(uv + 0.5 * SUMMIT_XL_D_TRACKS_M * uw); // sign according to urdf (if wheel
model is not symetric, should be inverted)

// Motor control actions
double limit = 40.0;

//ROS_INFO("epv=%5.2f, epw=%5.2f *** dUl=%5.2f dUr=%5.2f", epv, epw, dUl, dUr);

joint_state_blw_->commanded_effort_ = saturation(-10.0 * (joint_state_blw_->velocity_ - dUl), -limit, limit);
joint_state_flw_->commanded_effort_ = saturation(-10.0 * (joint_state_flw_->velocity_ - dUl), -limit, limit);
joint_state_brw_->commanded_effort_ = saturation(-10.0 * (joint_state_brw_->velocity_ - dUr), -limit, limit);
joint_state_frw_->commanded_effort_ = saturation(-10.0 * (joint_state_frw_->velocity_ - dUr), -limit, limit);
}

/// Controller stopping in realtime
void SummitXLControllerClass::stopping()
{



// Set the base velocity command
void SummitXLControllerClass::setCommand(const geometry_msgs::Twist &cmd_vel)
{
    v_ref_ = saturation(cmd_vel.linear.x, -10.0, 10.0);
    w_ref_ = saturation(cmd_vel.angular.z, -1.0, 1.0); // -10.0 10.0
}

```

```

}

void SummitXLControllerClass::commandCallback(const geometry_msgs::TwistConstPtr& msg)
{
    base_vel_msg_ = *msg;
    this->setCommand(base_vel_msg_);
}

double SummitXLControllerClass::saturation(double u, double min, double max)
{
    if (u>max) u=max;
    if (u<min) u=min;
    return u;
}

} // namespace

/// Register controller to pluginlib

PLUGINLIB_DECLARE_CLASS(summit_xl_ctrl, SummitXLControllerPlugin,
    summit_xl_ctrl_ns::SummitXLControllerClass,
    pr2_controller_interface::Controller)

```

10 Descripción del controlador del summit XL

A continuación nos proponemos a explicar el software controlador del robot. El cual corresponde a un controlador de un robot diferencial común. El control esta expresado en el lenguaje de programación C++.

```

{
// 4-Axis Skid Steer Rover

```

Se procede a determinar los parámetros iniciales del robot, en este caso se considera las velocidades del robot, velocidad lineal y velocidad angular. Las cuales son dadas cero, para una posición inicial del robot.

```

// Robot Speeds
linearSpeedMps_ = 0.0;
angularSpeedRads_ = 0.0;

// Robot Positions
robot_pose_px_ = 0.0;
robot_pose_py_ = 0.0;
robot_pose_pa_ = 0.0;

```

```

// External speed references
v_ref_ = 0.0;
w_ref_ = 0.0;

// Frequency
frequency = 100.0; // Hz 50.0

// for later access
node_ = n;

cmd_sub_ = node_.subscribe<geometry_msgs::Twist>("command", 1,
&SummitXLControllerClass::commandCallback, this);

return true;
}

/// Controller startup in realtime
void SummitXLControllerClass::starting()
{
// Initial positions
init_pos_blw_ = joint_state_blw_->position_;
init_pos_flw_ = joint_state_flw_->position_;
init_pos_brw_ = joint_state_brw_->position_;
init_pos_frw_ = joint_state_frw_->position_;
// Initial velocities
init_vel_blw_ = joint_state_blw_->velocity_;
init_vel_flw_ = joint_state_flw_->velocity_;
init_vel_brw_ = joint_state_brw_->velocity_;
init_vel_frw_ = joint_state_frw_->velocity_;

}

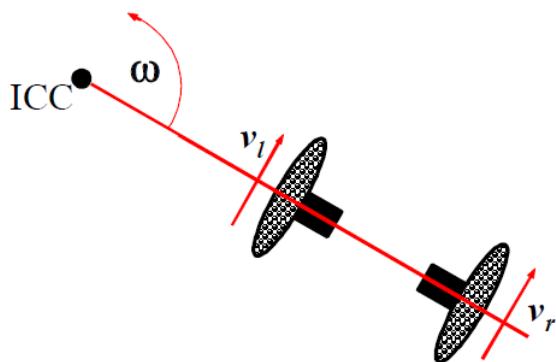
```

A continuación se procede con la lógica del movimiento del robot, basado en formulas y ecuaciones del modelo cinemático del robot

```
/// Controller update loop in realtime
void SummitXLControllerClass::update()
{
    double v_left_mps, v_right_mps;
```

Se calcula la velocidad de los dos pares de ruedas del robot, primer instancia se calcula la velocidad del par izquierdo de ruedas, la cual viene dada por la suma de ambas entre dos, para lograr un promedio de ambas velocidades, debido a que esta velocidad está dada en función de radianes, debe ser multiplicada por el radio de la rueda, dado en este caso por el diámetro de la rueda (previamente establecido SUMMIT_XL_WHEEL_DIAMETER) entre dos.

El principio de esta fórmula puede ser explicado de la siguiente forma:



Configuración de las ruedas de un robot diferencial

La velocidad de la rueda viene dada por $v = 2\pi r/T$, donde T es el tiempo que tarda la rueda en dar una vuelta sobre ICC el cual es el CENTRO INSTANTANEO DE CURVATURA, y siendo la velocidad angular $w = 2\pi/T$. al igualar estas dos ecuaciones podemos obtener que $w = 2\pi/T = 2\pi r/rT = v/r$, lo cual se puede obtener: $wr=v$

```
// Calculate its own velocities for realize the motor control
v_left_mps = ((joint_state_blw_->velocity_ + joint_state_flw_->velocity_) / 2.0) *
(SUMMIT_XL_WHEEL_DIAMETER / 2.0);
```

Se repite el procedimiento para el par derecho de ruedas.

```
v_right_mps = -((joint_state_brw_->velocity_ + joint_state_frw_->velocity_) / 2.0) *
(SUMMIT_XL_WHEEL_DIAMETER / 2.0); // sign according to urdf (if wheel model is not
symetric, should be inverted)
```

Una vez se obtiene la velocidad de cada lado de ruedas del robot, el controlador se dispone a obtener las velocidades del robot, la cual es el resultado directo de la relación entre las velocidades de ambos pares de ruedas.

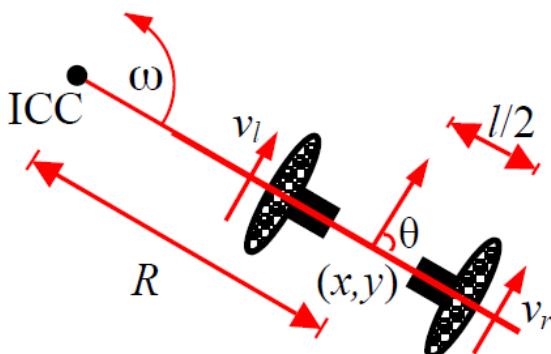
En el caso de la velocidad lineal del robot, la velocidad viene dada como una relación lineal entre ambas velocidades, siendo la misma el promedio de ambas.

```
linearSpeedMps_ = (v_right_mps + v_left_mps) / 2.0; // m/s
```

Nótese, que en el paso anterior se obtuvieron las velocidades para el lado derecho e izquierdo del robot, basados en la misma velocidad angular. A partir de eso podemos decir que:

$$\begin{aligned} w(R+l/2) &= v_r \\ w(R-l/2) &= v_l \end{aligned}$$

Donde, R es la distancia entre el centro de curvatura y el centro del robot, y l es la distancia entre ambos pares de ruedas expresado en nuestro caso como `SUMMIT_XL_D_TRACKS_M`, como se explica en la figura siguiente.



Esquema de las ruedas del robot.

Resolviendo en función de la velocidad angular obtenemos:

$$\begin{aligned} R &= (l/2)(v_l + v_r) / (v_r - v_l) \\ W &= (v_r - v_l) / l \end{aligned}$$

```
angularSpeedRads_ = (v_right_mps - v_left_mps) / SUMMIT_XL_D_TRACKS_M; // rad/s
```

```
//ROS_INFO("linearSpeedMps=%5.2f, angularSpeedRads=%5.2f", linearSpeedMps_, angularSpeedRads_);
```

```
//Controlador PD
```

El controlador derivativo se opone a desviaciones de la señal de entrada, con una respuesta que es proporcional a la rapidez con que se producen éstas.

```
// Current AX3500 controllers close this loop allowing (v,w) references.
```

```

double epv=0.0;
double epw=0.0;
static double epvant =0.0;
static double epwant =0.0;

```

El control proporcional es el tipo de control que utilizan la mayoría de los controladores que regulan la velocidad de robot. Si el robot se encuentra moviéndose a la velocidad objetivo y la velocidad aumenta ligeramente, la potencia se reduce ligeramente, o en proporción al error (la diferencia entre la velocidad real y la velocidad objetivo), de modo que el automóvil reduce la velocidad poco a poco y la velocidad se aproxima a la velocidad objetivo, por lo que el resultado es un control mucho más suave que el control tipo encendido/apagado.

En este caso kpv y kpw serán las ganancias proporcionales para el controlador PD, para la velocidad lineal y angular, respectivamente.

```

// Adjusted for soft indoor office soil
double kpv=10.0; double kdv=0.0;
double kpw=20.0; double kdw=0.0;

```

En el algoritmo de control proporcional, la salida del controlador es proporcional a la señal de error, que es la diferencia entre el punto objetivo que se desea y la variable de proceso. En otras palabras, la salida de un controlador proporcional es el producto de la multiplicación de la señal de error y la ganancia proporcional.

Esto puede ser expresado matemáticamente:

$$P_{\text{out}} = K_p e(t)$$

Donde

- P_{out} : Salida del controlador proporcional
- K_p : Ganancia proporcional
- $e(t)$: Error de proceso instantáneo en el tiempo t .

$$e(t) = SP - PV$$
- SP : Punto establecido
- PV : Proceso variable

De esta forma nos proponemos a encontrar el error de proceso instantáneo, para las velocidades lineales y angulares.

```

// State feedback error
epv = v_ref_ - linearSpeedMps_;
epw = w_ref_ - angularSpeedRads_;

```

En este punto aplicamos la acción de control proporcional-derivativa, la cual se define por la fórmula:

$$u(t) = K_p e(t) + K_p T_d \frac{de(t)}{dt}$$

Donde Td es una constante denominada tiempo derivativo.

```
// Compute state control actions
double uv= kpv * epv + kdv * (epv - epvant);
double uw= kpw * epw + kdw * (epw - epwant);
epvant = epv;
epwant = epw;
```

Esta acción tiene carácter de previsión, lo cual hace más rápida la acción de control, aunque tiene la desventaja importante que amplifica las señales de ruido y puede provocar saturación en el actuador.

```
// Inverse kinematics
double dUI = uv - 0.5 * SUMMIT_XL_D_TRACKS_M * uw;
double dUr = -(uv + 0.5 * SUMMIT_XL_D_TRACKS_M * uw); // sign according to urdf (if wheel
model is not symetric, should be inverted)

// Motor control actions
double limit = 40.0;

//ROS_INFO("epv=%5.2f, epw=%5.2f *** dUI=%5.2f dUr=%5.2f", epv, epw, dUI, dUr);

joint_state_blw_->commanded_effort_ = saturation(-10.0 * (joint_state_blw_->velocity_ -
dUI), -limit, limit);
joint_state_flw_->commanded_effort_ = saturation(-10.0 * (joint_state_flw_->velocity_ -
dUI), -limit, limit);
joint_state_brw_->commanded_effort_ = saturation(-10.0 * (joint_state_brw_->velocity_ -
dUr), -limit, limit);
joint_state_frw_->commanded_effort_ = saturation(-10.0 * (joint_state_frw_->velocity_ -
dUr), -limit, limit);
}
```

11 Conclusiones

Durante el proceso de modelado y programación del summit XL, se hizo uso extensivo de las herramientas aprendidas durante el curso de Diseño y Control de Robots. Aunque los algoritmos y modelos presentados en este informe fueron realizados con los mínimos requerimientos de complejidad, con el fin de simplificar el trabajo, y acelerar el proceso de modelado y programación del summit XL. Se pudo comprobar, que estos modelos se aproximan con seguridad al comportamiento real del robot. Sin embargo el propósito de los mismos, es brindarle al lector una introducción al complejo mundo de la robótica, y al uso de herramientas como ROS, ya que nos es muy intuitivo, sino se disponen de conocimientos en telecomunicaciones, programación, y sistemas operativos basados en Linux. También, brinda una idea de cómo, se realizan diseños en robótica, empezando por modelos en papel (dinámicos y cinemáticos), luego traduciéndolos a líneas de códigos de software de simulación (en nuestro caso MATLAB), el cual mediante el análisis de datos y gráficas, nos da una aproximación del comportamiento que tendrá el robot. Una vez se comprueba el

funcionamiento del algoritmo de control, se traduce el código al sistema ROS. Luego, se carga el algoritmo de control al robot, y se comprueban los resultados, ajustando así los parámetros de control hasta alcanzar el comportamiento, y desempeño deseado. El desarrollo de este proyecto, complementa lo aprendido en clases, y abre las puertas a futuros desarrollos más complejos y avanzados.

12 Referencias

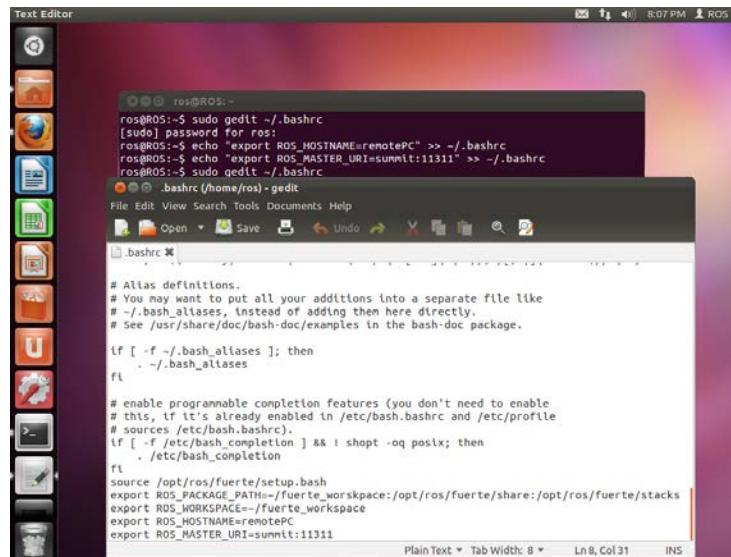
- <http://www.cs.columbia.edu/~allen/F11/NOTES/icckinematics.pdf>
- <http://www8.cs.umu.se/research/uminf/reports/2011/019/part1.pdf>
- Robotics, vision and Control – Fundamental Algorithms in MATLAB (Springer Tracts in Advanced Robotics).pdf- Peter Corke.
- Modeling and Control of a 4-Wheel Skid-Steering Mobile Robot - Krzysztof Kozłowski, Dariusz Pazderski.
- Summit XL Mobile Robot - System Architecture Manual – Version 1.1
- www.ros.org

APÉNDICE

A.1. Variables De Entorno De ROS

Opcionalmente, si desea una buena costumbre añadir al .bashrc las variables de entorno de ajuste de red de ROS. Para esto, realizamos esta acción primeramente en la maquina principal:

```
echo "export ROS_HOSTNAME=remotePC" >> ~/.bashrc
echo "export ROS_MASTER_URI=summit:11311" >> ~/.bashrc
. ~/.bashrc
```



NOTA: Recomendamos primero probar bajo este tutorial la variable ROS_MASTER_URI con 'localhost:11311'.

Para entender más de variables de entorno visitar:

<http://www.ros.org/wiki/ROS/EnvironmentVariables>

Para entender más acerca de la variable de entorno de stack visitar:

http://www.ros.org/wiki/ROS/EnvironmentVariables#ROS_PACKAGE_PATH

A.2. SUMMIT XL Troubleshooting

1. ***EL CONTROL BLUETOOTH NO ENLAZA CON EL ROBOT Y NO PUEDE SER CONTROLADO.***

En el caso particular de este problema se da cuando se apaga por descarga de batería. Cuando se inicia el sistema operativo del robot, probablemente quede en la pantalla de GRUB esperando a que se seleccione una opción. Seleccione la opción de booteo conectando un teclado USB y presionando ENTER.

Adicionalmente:

- Si desea puede conectar un monitor y navegar con nautilus el sistema de archivos.
- La contraseña de ingreso es "summit", sin doble comillas.

2. ***EL PAQUETE DESCARGADO DEL REPOSITORIO DE SUMMIT NO PUEDE SER COMPILADO***

Esto se debe a la variable de entorno \$ROS_PACKAGE_PATH.

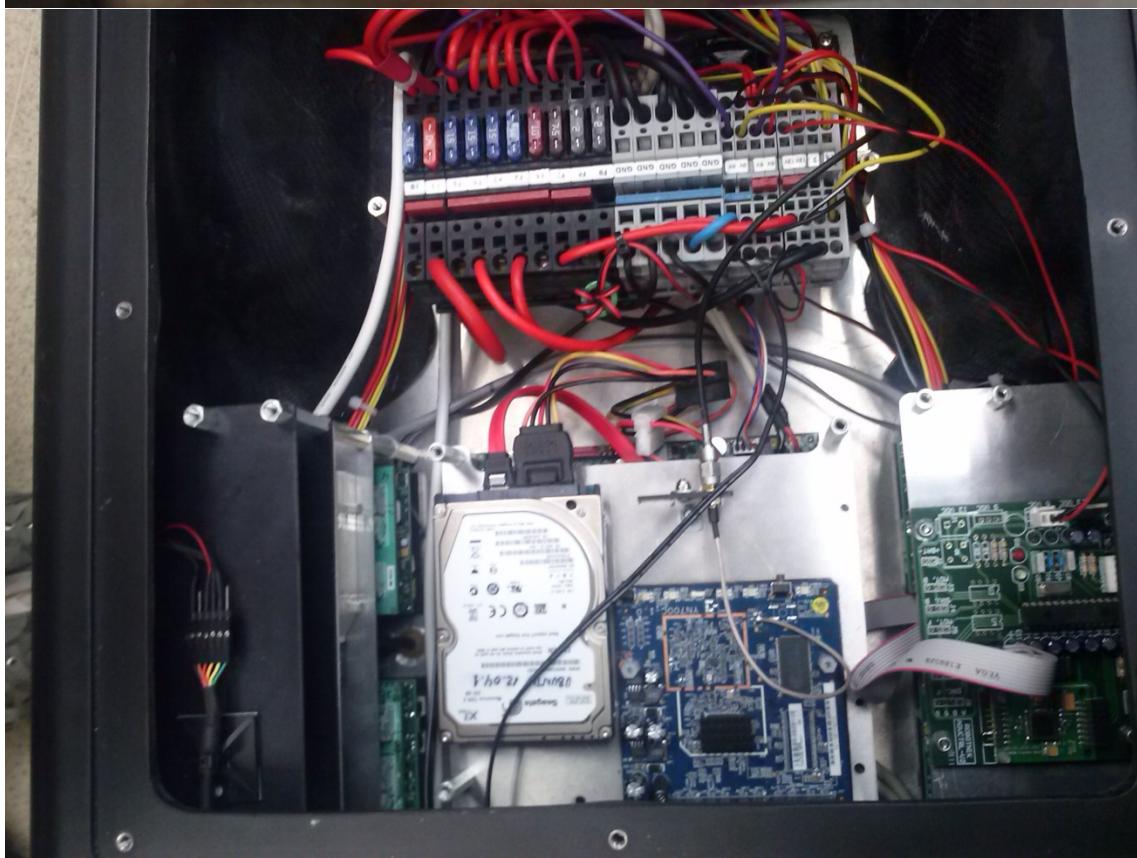
La manera correcta será:

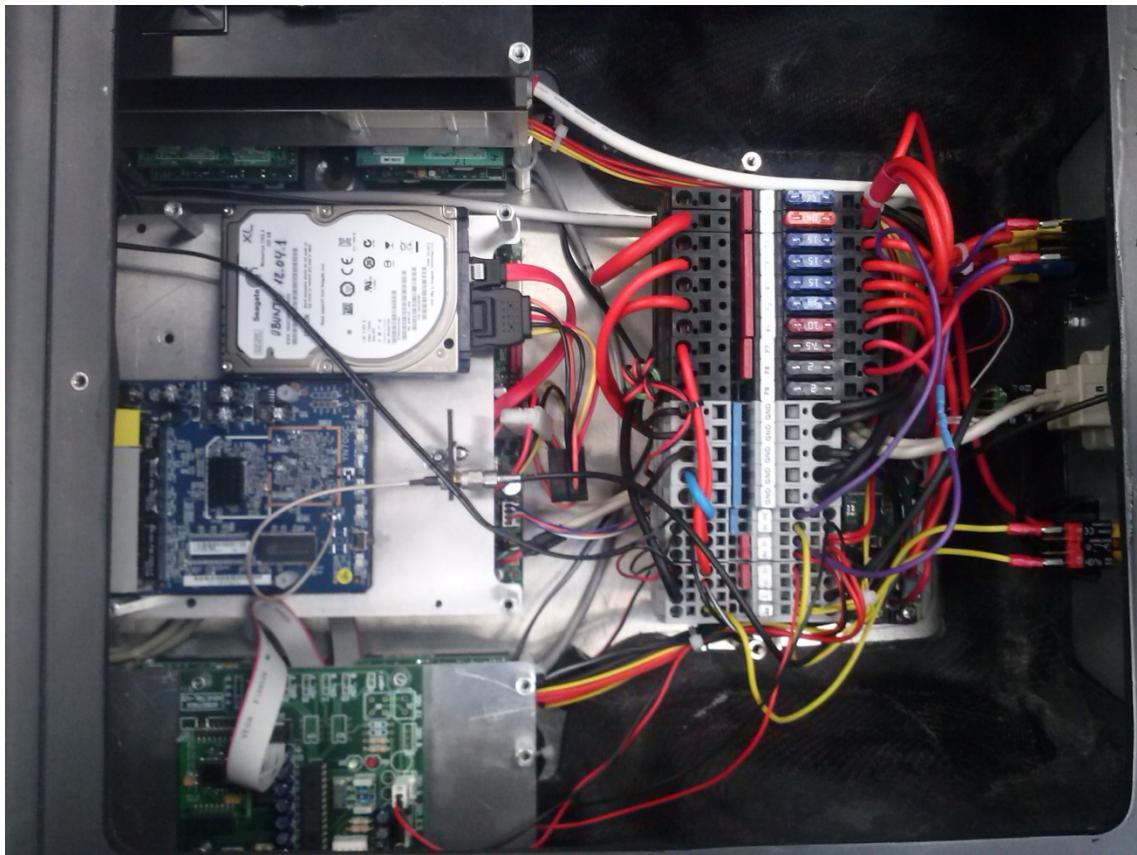
- Descargar el stack de ROS en el espacio de trabajo (asumiendo que es \$ROS_WORKSPACE=~/fuerte_workspace) y dentro de una carpeta... svn checkout [http://summit-xl-ros-stack.googlecode.com/svn/trunk/trunk/summit_xl_sim_fuerte\(summit-xl\)](http://summit-xl-ros-stack.googlecode.com/svn/trunk/trunk/summit_xl_sim_fuerte(summit-xl))
- Ejecutar 'sudo gedit ~/.bashrc'
- Modificar \$ROS_PACKAGE_PATH hasta la carpeta donde está el stack \$ROS_PACKAGE_PATH=~/fuerte_workspace/summit-xl:\$ROS_PACKAGE_PATH
- Guardar y cerrar gedit
- Ejecutar '. ~/bashrc' para que tome los cambios
- Finalmente verificar si todo está bien, con el comando rospack puede encontrar el paquete summit_xl_ctrl que es uno de los paquetes descargados. 'rospack find summit_xl_ctrl'

Para formatear una memoria USB bajo entorno gráfico, seguir el tutorial:

<http://kuyne.blogspot.com/2012/05/como-formatear-una-memoria-usb-en.html>

A.3 IMÁGENES







~ 80 ~