Aversivo / servo Microb 2008

Un WikiDroids artículo.

Documentación esclavitud Microb Tecnología 2008, Oliver Matz, Fabrice Desclaux (evitación de obstáculos)

- *Esta documentación está aún en elaboración, que cambia regularmente. No dude en reportar cualquier incoherencias, o cualquier punto que no se aborda con suficiente claridad. Mi correo electrónico: zer0@droids-corp.org*
- *Los ejemplos presentados aquí son sólo para fines didácticos, es posible que ni siquiera compilar (si es mejor referirse al código de nuestro robot)*

Resumen

Introducción

Principio básico de la esclavitud

Un servo es un sistema de bucle para el control de un proceso que tiene una entrada y una salida. El propósito de este documento es presentar una solución para la implementación de un módulo de control digital con aversión , aplicadas al control de un robot para Eurobot, pero que podría ser utilizado para otras aplicaciones.

Este control se lleva a cabo en un microcontrolador Atmel AVR, y ha sido utilizado en 2007 (con algunas variaciones) por los

equipos de Eirbot, Tecnología Microb, ESIAL Robótica y Maggie. Es relativamente bien adaptado a otro robot y todo el código está disponible bajo la licencia GPL.

Después de la presentación de la mecánica y la electrónica del robot, podemos discutir el código de una perspectiva global. Luego a través de varios ejemplos cuya dificultad aumenta progresivamente a medida que añadimos uno a uno los ladrillos de nuestra esclavitud.

A poca documentación sobre el control digital

- Servo ACVR (http://www.robot-cva.com/?page=tech) (IUT Ville d'Avray), la documentación de 2006, un poco corto, pero se describen los principios de la esclavitud de los mejores robots de corte
- Descripción del servo robot ClubElek del INSA de Lyon 2007 (http://clubelek.insa-lyon.fr/joomla/fr/base_de_connaissances/informatique/asservissement_et_pilotage_de_robot_autonome_introduc_2.php) , bien escrito y bien ilustrado
- La esclavitud de UTC (http://wwwassos.utc.fr/coupem6/2007/fichiers/RapportTX.pdf) , el informe de 2006.
- Esclavitud Microb Tecnología 2005 (http://microb.technology.free.fr/fr/asservissement2005/index.html) , el precursor del documento que está leyendo
- Servo Robot Eirbot (ENSEIRB) en 2003 (http://f4eru.free.fr/robot/asserv03.doc) , el antepasado del antepasado
- La esclavitud de 2007 Eirbot nuestros amigos (http://eirbot.enseirb.fr/documents/DocAsserv2k7.pdf) que también usan aversivo.
- Un hilo del foro del Planeta-Ciencia (http://www.planete-sciences.org/forums/viewtopic.php?t=8571&start=0&postdays=0&postorder=asc&highlight=) en el que se discuten muchos puntos, incluyendo Gargamel, un ex profesor de la IUT de VA, y experto en la materia.
- Conducir en la wikipedia (http://fr.wikipedia.org/wiki/Asservissement)
- Implementación de un controlador digital
- Añade que enlaza aquí

Especificación de nuestra assservissement

La aplicación debe cumplir con las siguientes limitaciones:

De usos múltiples : puede esclavizar a los diferentes sistemas simples con un PID, u otro filtro. Las funciones de entrada y salida no son necesariamente los codificadores incrementales y un PWM como un robot (por ejemplo, calefacción, motor sin escobillas, ...)

Modular : permite una gestión totalmente independiente realizada bajo o manual de gestión de la interrupción en un bucle.

Laptop : Abril en la mayoría de (pero conservando la capacidad de ejecutar la mayoría de las cosas en un PC).

Dinámica : debemos ser capaces de crear instancias de un servo en cualquier momento, y quitar cuando ya no sea necesario. Por otra parte, debería ser posible para manejar servo múltiples a la vez, dentro de los límites de la memoria y el poder UC, por supuesto.

Especificidades para un robot Eurobot

- Posición de control
- servo rueda diferencial, que controla el ángulo y la distancia del robot en lugar de ruedas cada uno independientemente.
- Capacidad de generar trayectorias con la curva de velocidad trapezoidal (velocidad máxima y aceleración, la anticipación de más de frenado para parar en la posición correcta).
- La localización del robot en el campo: x, y, a.
- Interruptores rotativos, además de no motorizados de ruedas motorizada.
- generador de trayectoria para evitar los obstáculos y puntos de control.

Mecánica de nuestro robot

Dos ruedas motorizadas en la parte posterior, frontal con bola

Nuestro robot está cerca de la mayoría mecánica del robot gran parte de la corte. Cada uno de los dos motores en el control de la rueda trasera.

Las ruedas de los codificadores de 2006 y su robot por separado

El bloque del motor de 2006

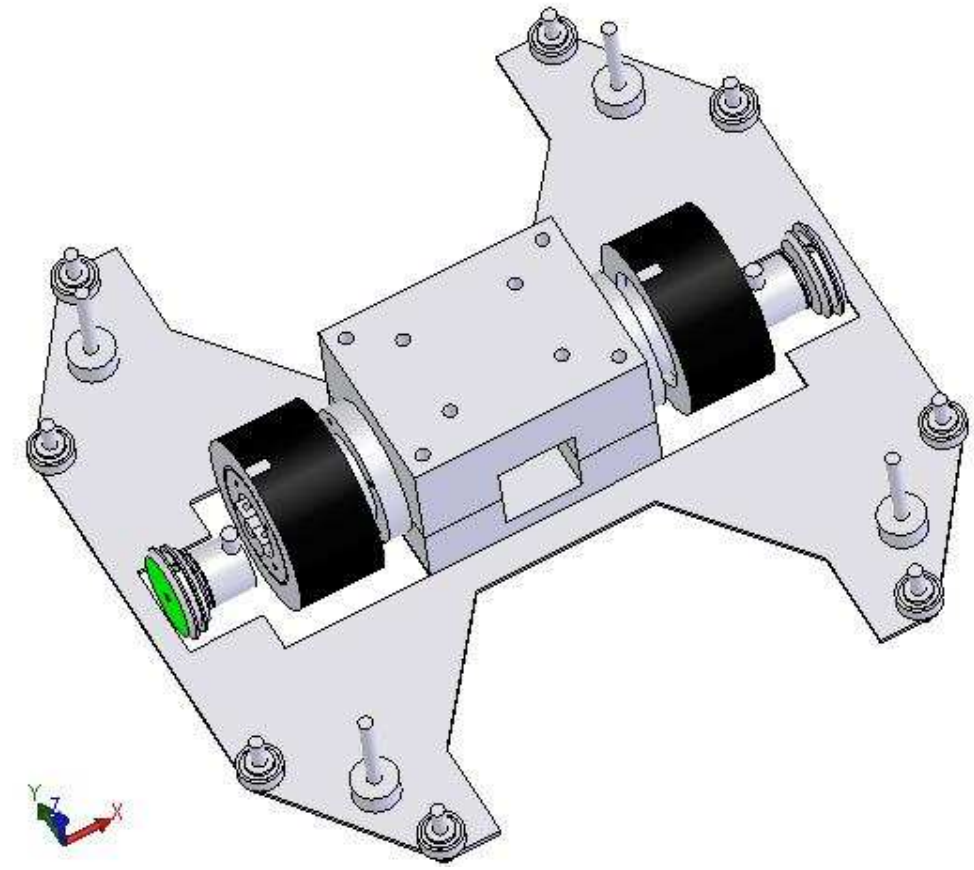
El robot Microb Tecnología 2006, de las siguientes

Ingeniería de precisión

Es necesario que el mecánico se realiza con tanta precisión como sea posible. De hecho, un simple juego en el cuentakilómetros de fijación no-motorizados provoca una desviación en la medición de la posición del robot de hacer cualquier ubicación muy difícil. Del mismo modo, la distribución del peso sobre los distintos puntos de apoyo es muy importante: el peso máximo debe ser de ruedas motorizada. Con un código equivalente en 2006 y 2007, el comportamiento del robot era mucho mejor en 2007 gracias a una mecánica mejor se dio cuenta.

Codificadores separados

Como se indicó anteriormente, el robot tiene, además de ruedas para la propulsión del robot, haciendo que la posición de la rueda de medición no motorizados. De hecho, las ruedas motorizada, para ser eficaz en la adhesión, debe ser bastante grande y suave. Desgraciadamente, éste es incompatible con una medida fiable de la posición: el diámetro de las ruedas y la distancia entre dos ruedas impreciso y apisonamiento del neumático en las curvas. Por otra parte, estas ruedas también tienden a deslizarse cuando el bloqueo contra un obstáculo o una fuerte aceleración.



El uso de codificadores externos guarda en la medición de precisión de la posición, y también detectar patinaje (el valor del encoder del motor es entonces decorrelated de la de los codificadores externos).

Cabe señalar que, contrariamente a nuestra elección en 2006, parece preferible colocar los interruptores sin motor rotatorio en el exterior y no interior, como lo hicimos en 2007.

Electrónica del robot

2006 placa base

2006 Power Board

Usted puede ver los detalles a continuación:

- Microb_Technology/2006/electronics
- Microb_Technology/2007/electronics

En resumen, el microcontrolador central genera un PWM y un poco de sentido para cada motor para funcionar. Usamos motores de corriente continua, controlada por el H-puente (LMD18200T). El valor de cada odómetro que viene en forma de contadores grises de 2 bits (el famoso Un canales A y B de los interruptores rotativos) es descifrado por los más pequeños

microcontroladores (AT90S2313) que lo transforma en un niño de 8 bits. Todos los cálculos se realizan en el núcleo del microcontrolador (ATMega128), no usamos componentes especializados como el LM629.

Tenga en cuenta que otros equipos como Eirbot ESIAL Robotik o utilizar una tarjeta diferente, pero sólo la configuración y los codificadores módulo de lectura debe ser cambiado (en Eirbot, la decodificación se realiza por un CPLD que gestiona cuatro codificadores).

Especificidades de nuestra esclavitud

Servo-Alpha Theta

Otra opción es la de la servo-Alpha Theta. Experimentamos con Eirbot en 2004 y desde 2005 por la Tecnología y Microb Eirbot. No es para esclavizar, como en general en los robots de corte, cada una de las ruedas con un PID, pero en lugar de usar un PID para esclavizar a la diferencia de posiciones de la rueda (ángulo alfa), y otro para la EPI esclavizar a la distancia (Theta).

Un ejemplo sencillo es el límite de tiempo: en este caso, n'asservit esa distancia y no el ángulo, lo que permite que el robot no oponer resistencia al ángulo de registro. Por otra parte, intuitivamente, aunque qu'asservir ONSENTIMIENTO un ángulo y distancia más larga corresponde a esclavizar a los robots en lugar de cuando esclavizados cada rueda de forma independiente. Para colmo, vemos que este tipo de control facilitará nuestro trabajo para la generación de trayectoria.

Descripción general del programa final

Aversivo

Nuestro código se basará en la propuesta de aversión que desarrollar un marco para la elaboración de GPL microcontrolador AVR con GCC y libc abril- El objetivo de este proyecto es proporcionar a las bibliotecas a utilizar las funciones de AVR de modo sencillo. Aversivo introduce el módulo , es un programa compilado como una biblioteca y proporcionar una característica. Por ejemplo, el módulo de PWM para configurar los registros internos para generar la señal PWM y proporcionar la interfaz de usuario fácil de entender.

El código forma parte de nuestra esclavitud tanto, aversión a la forma de varios módulos, cada uno con un propósito específico. Ya podemos considerar cuatro grupos funcionales: una tiene como objetivo el sometimiento, por ejemplo, el control de la velocidad de un motor. En segundo lugar, tenemos que calcular nuestra posición actual en x, y, un campo, y en tercer lugar, debemos proporcionar una interfaz para la generación de trayectorias de alto nivel para nuestro robot. Por último, el grupo funcional último será responsable de supervisar el buen funcionamiento del conjunto y para detectar posibles cuellos de botella, patinaje, ...

Sistema de control

- dispositivos / control_system / control_system_manager : Este es el módulo principal de la esclavitud, que gestiona las llamadas de otros módulos.
- dispositivos / control_system / filtros / pid : Un filtro de tipo PID (proporcional, derivada, integral).
- dispositivos / control_system / filtros / rampa : filtro para limitar la derivada de la entrada, no se utiliza en el control de nuestra posición, pero puede estar en un control de velocidad.
- dispositivos / control_system / filtros / quadramp : Filtro que permite limitar la segunda derivada y derivada de la entrada. Este es el filtro que genera curvas de velocidad de forma trapezoidal.

Posición

- dispositivos / robot / position_manager : calcular la posición del robot basada en codificadores.
- dispositivos / robot / robot_system : Definición de un proceso que describe el robot (ángulo y distancia). El módulo realiza las transformaciones necesarias para el paso de la marca (roueG, Rouède) a (ángulo, dist), y también administra

el hecho de que el robot cuenta con 4 codificadores.

Trayectoria

- dispositivos / robot / trajectory_manager : Definición de trayectorias de alto nivel para el usuario
- dispositivos / robot / obstacle_avoidance : Módulo para generar trayectorias con puntos de interés y evitar obstáculos.

Detección de bloqueo

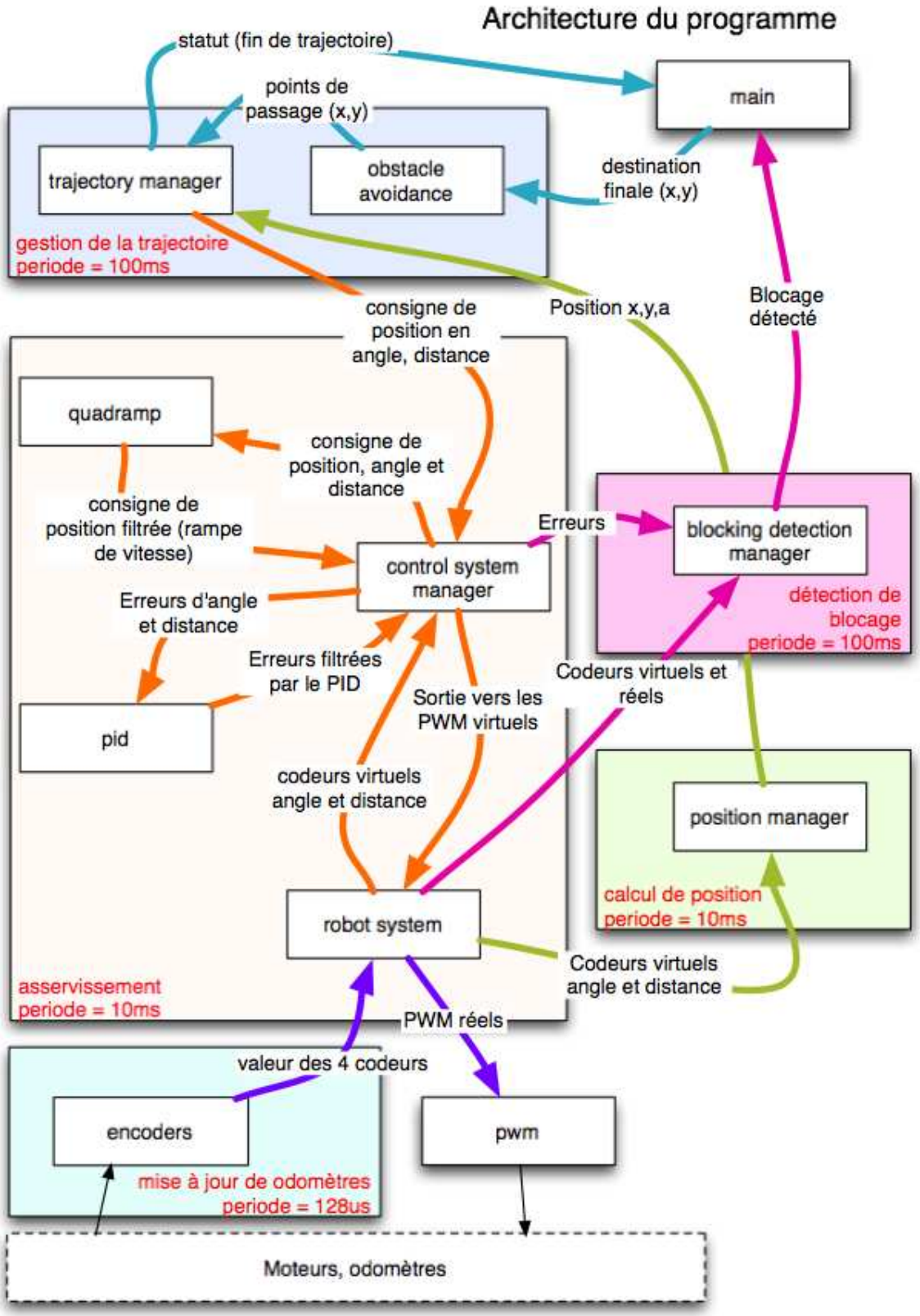
- dispositivos / robot / blocking_detection_manager : detección de interbloqueos en la esclavitud

Otros módulos, assservissement no específicos

- base / programador : para planificar la llamada de función a las prioridades de gestión.
- base/math/vect2 : las operaciones sobre vectores en 2D.
- dispositivos / codificadores / encoders_microb : gestión del codificador, nuestro propio hardware.
- de hardware / PWM : emisión de PWM para control de motores.

Diagrama de la final del servo

Finalmente, la esclavitud se organiza de la siguiente manera:



- 0 / El módulo principal del programa envía un comando a la trayectoria para llegar a un punto (x, y, a) de campo
- 1 / En base a la posición actual, el módulo de trayectoria le dará instrucciones adicionales para el ángulo de posición del servo y la distancia. Este sistema es actualizado periódicamente por un evento (período de alrededor de 100ms).
- 2 / El módulo se ejecuta en robot_system interrumpida periódicamente (de 5 a 10 ms), lee los codificadores valor real y actualizaciones de los valores de los sensores de ángulo y distancia (virtual).
- 3 / Por cada servo (ángulo y distancia), el módulo (control_system_manager CSM) envía la posición quadramp nuevo filtro
- 4 / El filtro devuelve el valor de consigna filtrada que limita la señal de derivados de derivados y en segundo lugar
- 5 / A continuación, las solicitudes MSC el valor de los codificadores virtual para determinar el ángulo y la distancia de error

- 6 / Estos errores se envían al filtro PID
- 7 / Cada filtro devuelve el valor filtrado
- 8 / PID de salida (ángulo y distancia) son enviados al módulo que determina las órdenes robot_system PWM para las ruedas
- 9 / Estos comandos se envían al módulo de PWM que modifica las señales generadas por la UC
- 10 / Otro evento que se ejecuta periódicamente (período de aproximadamente 10 ms), este es el position_manager que determinará la posición del robot en el área de juego en función de los codificadores de valor y la posición x, y, una anterior
- 11 / Esta posición es por supuesto para la trayectoria exécuition módulo siguiente

¿Dónde encontrar el código

Usamos una rama de desarrollo de la aversión avance de fase en la rama principal. El código fuente está aquí: http://cvsweb.droids-corp.org/cgi-bin/viewcvs.cgi/aversive/?pathrev=b_zer0

Código 2006: http://cvsweb.droids-corp.org/cgi-bin/viewcvs.cgi/aversive_projects/microb2006/motherboard_atm128/

Código (en desarrollo) de 2008: http://cvsweb.droids-corp.org/cgi-bin/viewcvs.cgi/aversive_projects/microb2008/main

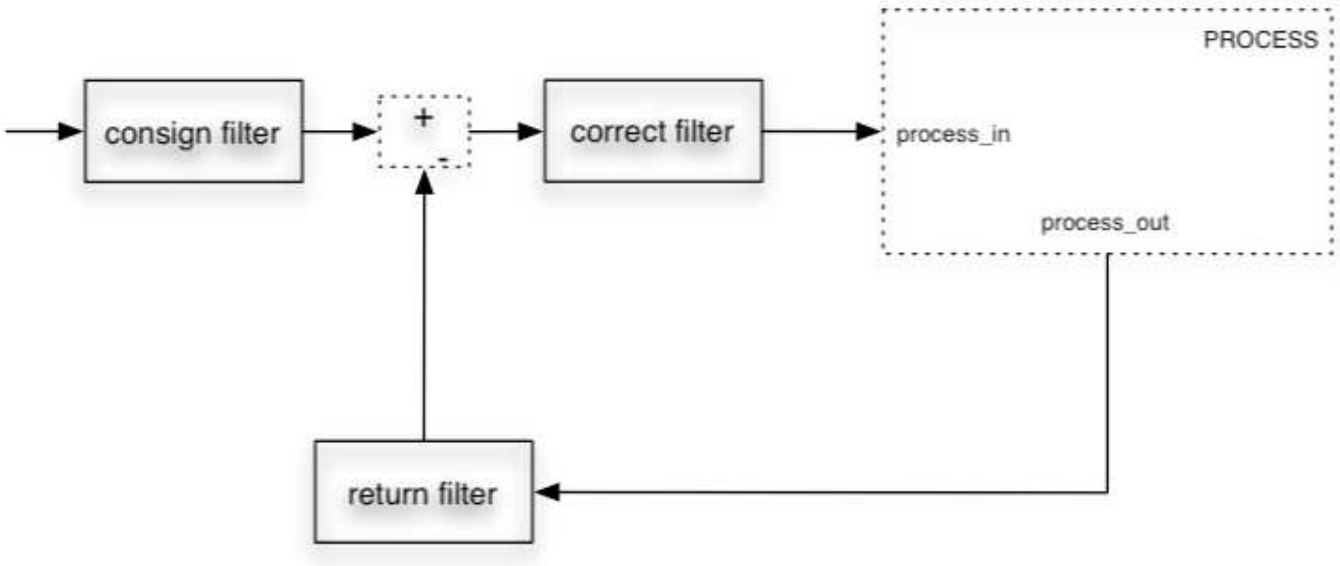
PID minimalista servo

Nuestro objetivo en esta sección es presentar dos módulos principales y pid control_system, que en conjunto permiten ya sólo un sistema de esclavitud. Vemos que este enfoque tiene muchos defectos serán corregidos en las siguientes secciones.

dispositivos / control_system / control_system_manager

Descripción

Este módulo se encarga de la esclavitud. Completamente solo, tiene poco uso, ya que requiere otros módulos que proporcionan diferentes filtros son necesarios para su funcionamiento, por ejemplo, el PID del módulo como el filtro de corrección. La estructura de datos CS contiene prácticamente sólo punteros a funciones y referencias a los metadatos propios filtros.



Estructuras

```
struct {CS int32_t (* consign_filter) (void *, int32_t); * consign_filter_params vacío, int32_t (* correct_fi
```

- Cada filtro se define mediante un puntero a la función de ejecutar y un puntero a los datos del filtro.
- consign_value contiene el conjunto actual.
- filter_consign_value contiene el conjunto actual después de que el conjunto de filtros.
- error_value contiene el error actual, es decir, la diferencia entre la referencia y se filtró dejando el proceso a controlar (resultado process_out function ()).
- out_value contiene la corriente de salida, es decir, el error después de pasar por el filtro de corrección. Este valor se envía a la entrada del proceso a través process_in ().

Interfaz

vacío cs_init (struct CS CS *)

Inicializa la estructura de CS: todos los campos se establecen en NULL. Es importante no olvidar este paso, porque cuando juegas con los punteros de función, no debe vincularse en cualquier lugar.

Suponemos que una función de filtro está inactiva cuando el puntero a la última es igual a NULL.

vacío cs_set_consign_filter (struct CS CS *, int32_t (* consign_filter) (void *, int32_t), consign_filter_params void *)

vacío cs_set_correct_filter (struct CS CS *, int32_t (* correct_filter) (void *, int32_t), correct_filter_params void *)

vacío cs_set_feedback_filter (struct CS CS *, int32_t (* feedback_filter) (void *, int32_t), feedback_filter_params void *)

vacío cs_set_process_in (struct CS CS *, void (* process_in) (void *, int32_t), process_in_params void *)

vacío cs_set_process_out (struct CS CS *, int32_t (* process_out) (void *), process_out_params void *)

Inicialización de las funciones y las funciones de filtro de entrada / salida del servo.

cs_do_process int32_t (struct CS CS *, consigna int32_t)

La llamada a la función cs_do_process int32_t (struct CS CS *, int32_t registrada) hace lo siguiente, a fin de:

- Seguridad de la instrucción en la estructura
- aplicación del filtro si la instrucción se define
- lee la salida del proceso (la salida es 0 si el proceso no se define - lazo abierto>)
- filtro feedback_filter si se pone en
- restando el proceso de punto de ajuste de salida filtrada filtrada
- error_value copia de seguridad
- aplicación del filtro de corrección si se define
- guarda el valor de salida
- envía la salida a process_in () si está definido.
- devuelve el valor de salida (el mismo que se envía al proceso)

vacío cs_manage (struct CS CS *)

Esta función simplemente llama a la cs_do_process function () con las instrucciones previamente almacenado en la estructura (un cs_set_consign (struct CS CS *, v int32_t), por ejemplo.

cs_get_out int32_t (struct CS CS *)

Devuelve la última salida enviado al proceso de

cs_get_error int32_t (struct CS CS *)

Devuelve el último error calculado

```
int32_t cs_get_consign (struct CS CS *)
```

Devuelve el conjunto actual

```
int32_t cs_get_filtered_consign (struct CS CS *)
```

Devuelve el registro actual filtrada

```
cs_set_consign vacío (struct CS CS *, v int32_t)
```

Modifica el sistema no se base en el cálculo de la esclavitud, a diferencia de `cs_do_process ()`.

dispositivos / control_system / filtros / pid

Descripción

Este módulo es un filtro, es decir, que implementa la interfaz se define en los filtros de doc .

Esto le permite aplicar un PID, es decir, la suma ponderada de la integral, derivados y la propia entrada. Nuestro filtro tiene unas pocas extensiones en comparación con un PID convencional, ya que también permite que algunas señales para saturar (entrada, a término, de salida).

Estructuras

```
struct {int16_t gain_P pid_filter / ** <* Ganancia de módulo proporcional / gain_I int16_t / ** <* Ganancia d
```

Interfaz

```
vacío pid_init (struct * p pid_filter);
```

Inicializa los valores de la estructura: una ganancia para el 0 proporcional, para los demás. Saturaciones son discapacitados, y el total divisor es 1. Campos y last_in Integral se establecen en 0. El filtro se comporta como la identidad del filtro (también llamado filtro-que-usa-para-nada) cuando se ha inicializado.

```
pid_do_filter int32_t (struct * p pid_filter, en int32_t);
```

Se aplica el filtro en el valor y devuelve el resultado.

```
vacío pid_set_gains (struct * p pid_filter, gp int16_t, int16_t gi, gd int16_t);

vacío pid_set_maximums (struct * p pid_filter, max_in int32_t, int32_t MAX_I, max_out int32_t);

vacío pid_set_out_shift (struct * p pid_filter, out_shift int16_t);
```

De acceso que van a configurar el filtro.

```
pid_get_gain_P int16_t (struct * p pid_filter);

pid_get_gain_I int16_t (struct * p pid_filter);

int16_t pid_get_gain_D (struct * p pid_filter);

pid_get_max_in int16_t (struct * p pid_filter);

pid_get_max_I int16_t (struct * p pid_filter);
```

```
pid_get_max_out int16_t (struct * p pid_filter);

pid_get_out_shift uint8_t (struct * p pid_filter);

pid_get_value_I int32_t (struct * p pid_filter);

pid_get_value_in int32_t (struct * p pid_filter);

int32_t pid_get_value_D (struct * p pid_filter);

pid_get_value_out int32_t (struct * p pid_filter);
```

Los captadores.

Ejemplo

Este ejemplo de una línea de puntuación está allí para comprender el funcionamiento de los módulos descritos. La idea es controlar la velocidad de la rueda con un PID. Obviamente, no es conveniente aplicar un control de este tipo, vamos a ver más ejemplos avanzados más adelante en este documento.

```
estructura cs cs; pid_filter estructura p / * funciones para definir en otra parte del programa * / motor_set
```

Hacer servidumbre auto (bajo interrupción)

El siguiente paso consiste en hacer girar el servo en el fondo. De hecho, el microcontrolador seguramente tendrá otras cosas que hacer que hacer espera activa. Para ello, usamos el módulo planificador.

base / planificador

Descripción

Controlador de llamada a la función de interrupción.

La idea es planificar las funciones de llamada en un periódico o una sola. También ayuda a controlar la prioridad de los acontecimientos y evita que la llamada recursiva de la función misma.

Su papel en el control es simple, puede hacer autónomo, lo que significa que se ejecutará en virtud de la interrupción, dejando el programa principal cuando el cálculo ha terminado.

Interfaz

```
vacío scheduler_init (void);
```

Módulo de inicialización

```
scheduler_add_single_event_priority int8_t (void (* f) (void *), void * data, période uint16_t,
la prioridad uint8_t);

scheduler_add_periodical_event_priority int8_t (void (* f) (void *), void * data, période
uint16_t, la prioridad uint8_t);

scheduler_add_single_event int8_t (void (* f) (void *), void * data, período uint16_t);

scheduler_add_periodical_event int8_t (void (* f) (void *), void * data, período uint16_t);
```

Añade un evento a la mesa. No puede ser periódica o única. La prioridad puede ser especificado entre 0 y 255 (el valor más alto es la más alta prioridad). La función será llamada cada programada *período "* con *datos* como un parámetro.

Cada una de las funciones para agregar caso devuelve un identificador único del evento, dijo.

```
scheduler_del_event int8_t (número int8_t);
```

Quita el elemento identificado por el *número* .

Ejemplo

```
# Definir 10000/SCHEDULER_UNIT PERIODO / * 10 ms * / # define PRIORIDAD 100 struct cs cs; pid_filter estructu
```

Utilizar una velocidad de rampa en un control de velocidad

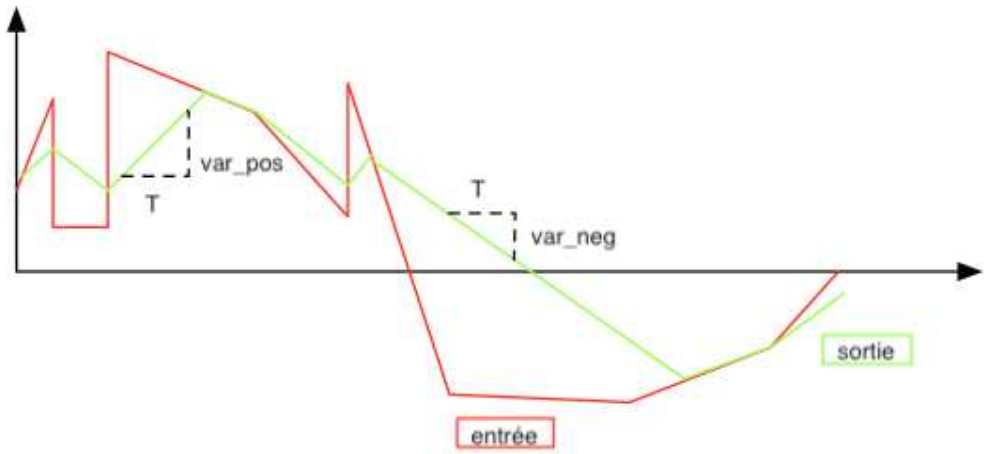
Ahora que hemos visto dos usos básicos de nuestro módulo de servo, vamos a tratar de evolucionar mediante la aplicación de un filtro en la entrada de la misma (el registro). Nuestro primer ejemplo es un control de velocidad. Si jugamos a cambiar de repente la configuración de velocidad, entonces es una apuesta segura que el PWM se enviará hasta alcanzar la velocidad deseada (con un ligero exceso o no, dependiendo de la configuración del PID).

Como es físicamente imposible para cambiar instantáneamente la velocidad de un motor, esto plantea un problema. De hecho, cuando el PWM es del maxium, la aceleración es determinado por las características del motor. Es tolerable si no hubiera un motor, pero si hay dos (uno para la rueda izquierda y el otro para la derecha), y no tienen exactamente la misma respuesta, entonces el robot no tendrá derecho durante la fase de aceleración. La solución a este problema es limitar la variación de los pedidos, y por lo tanto la aceleración.

control_system / filtros / rampa

Descripción

El filtro se impone una restricción en su entrada. Este módulo será responsable de limitar el conjunto de variación (su derivado).



Estructuras

```
struct {var_neg ramp_filter uint32_t; uint32_t var_pos, prev_out int32_t;}
```

var_neg var_pos y son, respectivamente, los cambios positivos y negativos máximos (en valor absoluto).

Interfaz

```
vacío ramp_init (struct r * ramp_filter);
```

Inicializa los valores de la estructura: var_pos y var_min a 0xFFFFFFFF. En este caso, el filtro actúa como un filtro de

copias que el de entrada.

```
ramp_do_filter int32_t (struct r * ramp_filter, en int32_t);
```

Se aplica el filtro en el valor y devuelve el resultado.

```
vacío ramp_set_vars (struct r * ramp_filter, neg uint32_t, pos uint32_t);
```

De acceso que van a configurar el filtro.

Ejemplo

```
estructura cs cs; estructura p pid_filter, estructura ra ramp_filter; # define 10000/SCHEDULER_UNIT PERIODO /
```

Cada vez que llame cs_manage (+ CS), el depósito se filtra desde la entrada definida por cs_set_consign ().

Para el ejemplo anterior, las instrucciones que valdrá la pena después de la filtración:

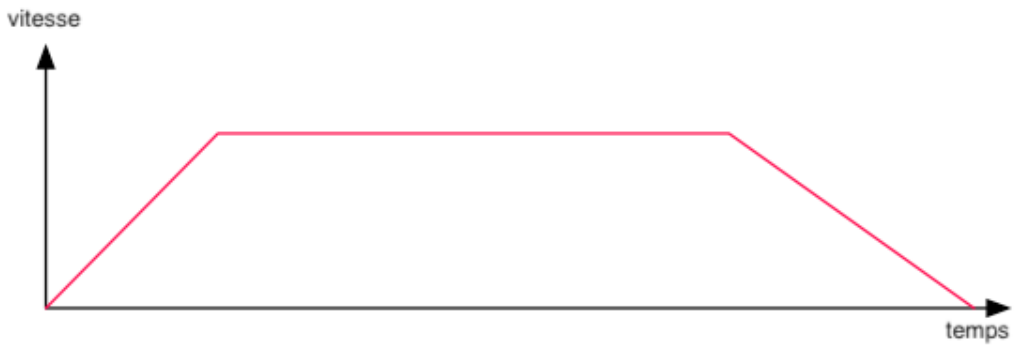
```
0, 0, 0, ... pdt un segundo ... 0, 2, 4, 6, 8, 10, 10, 10, ...
```

Si más adelante en el programa, llamado cs_set_consign (& r, -2), las instrucciones reales se encuentran de nuevo linealmente con una pendiente de -1:

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, -1, -2, -2, -2, ...
```

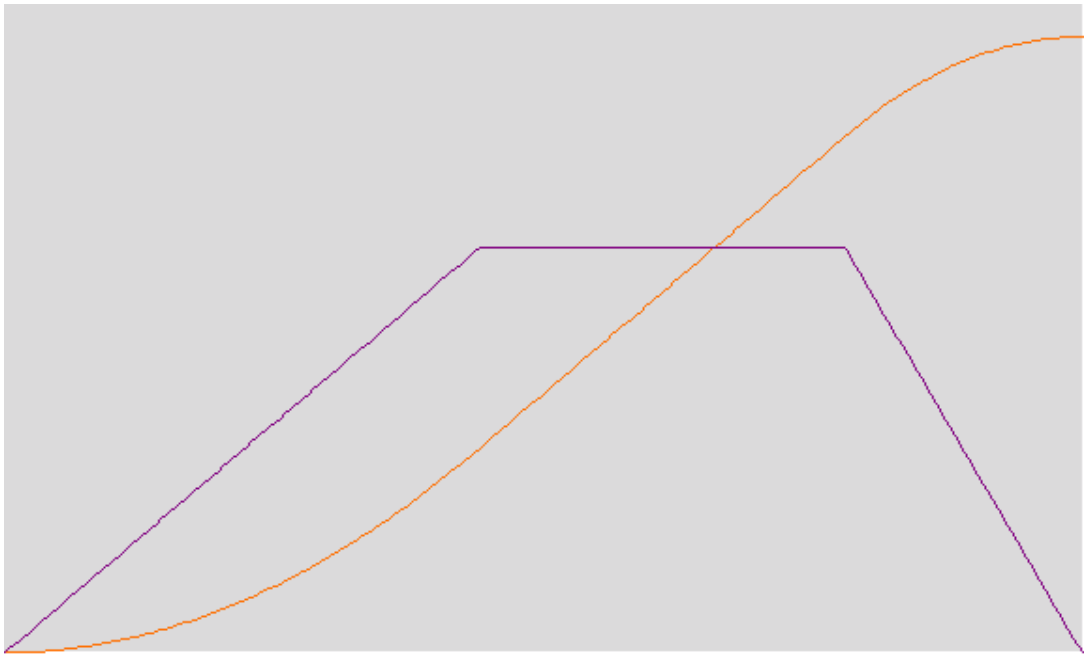
Una rampa de velocidad en un control de posición

Nos acerca a nuestro objetivo, es decir, un puesto de control donde se puede establecer un límite de aceleración / deceleración aaussi y velocidad. Queremos obtener una curva de velocidad de forma trapezoidal:



La diferencia con el ejemplo anterior es que no esclavizar a la velocidad, pero la posición. Esto significa que el recargo es la distancia total que el motor tiene que viajar de su cero. Si partimos de una zona trayectoria trapezoidal del trapecio debe ser igual a la posición de consigna.

Por ejemplo, el gráfico a continuación, el filtro de entrada es un paso (no mostrado). Esta entrada se corresponde con una instrucción posición deseada por el usuario. La salida es la instrucción posición de filtrado, suavizado por las restricciones que limitan la aceleración y la velocidad. Ella tiene una forma de "S" (naranja), y es la integral de la curva de V:



Orange, el comando de la posición que se desee. Púrpura, la velocidad alcanzada si se siguen estas instrucciones.

La dificultad de este filtro consiste en anticipar el momento en que debe comenzar a desacelerarse, por lo que la velocidad es cero cuando alcanza la posición deseada.

control_system / filtros / quadramp

Descripción

El tratamiento es mucho más complicado que en el caso de la rampa de filtro, ya que anticipan la desaceleración con el fin de llegar a la posición definida con velocidad cero.

Estructuras

```
struct {uint32_t var_2nd_ord_pos quadramp_filter; var_2nd_ord_neg uint32_t, var_1st_ord_pos uint32_t; uint32_
```

var_1st_ord_pos y var_1st_ord_neg son los límites de la primera derivada (*la velocidad* en el caso de un control de posición). var_2nd_ord_neg y var_2nd_ordpos son los límites de la segunda derivada (*aceleración* en el caso de un control de posición).

Interfaz

vacío quadramp_init (struct r * quadramp_filter);

> Inicializa los valores de la estructura: var_pos, var_min, derivate_var_pos, derivate_var_neg a 0xFFFFFFFF. En este caso, el filtro actúa como un filtro de copias que el de entrada.

quadramp_do_filter int32_t (struct r * quadramp_filter, en int32_t);

> Se aplica el filtro en el valor y devuelve el resultado.

vacío quadramp_set_2nd_order_vars (struct r * quadramp_filter, var_2nd_ord_pos uint32_t, var_2nd_ord_neg uint32_t);

vacío quadramp_set_1st_order_vars (struct r * quadramp_filter, var_1st_ord_pos uint32_t, var_1st_ord_neg uint32_t);

De acceso para establecer el filtro.

Ejemplo

Aceleración Aceleración = 1 max max = -1 Velocidad máxima = 10 minutos = 150 = posición -10

El filtro de salida es quadramp siguientes instrucciones P (denotamos su derivada V). Los valores son los siguientes (si V = 0 y P = 0 en t = 0):

```
V = 1,2,3,4, 5, 6, 7, 8, 9, 10,10,10,10,10, 10, 9, P = 1,3,6,10,15,21,28, 36,45,55,65,75,85,95,105,114,
```

```
V = 8, 7, 6, 5, 4, 3, 2, 1 P = 122.129.135.140.144.147.148.149.150
```

La derivada de la salida del filtro se corresponde con la velocidad establecida, y es de forma trapezoidal. La posición, por su parte, tiene forma de S: una parábola a la aceleración y desaceleración, y justo cuando la velocidad es estable.

La aplicación debe hacerse por completo con números enteros: aunque el uso del *doble* (= 32 bits en coma flotante con avr-gcc) en realidad simplificar el código mucho con la librería matemática, no es posible porque la la entrada es un bits y de doble conversión de 32 perdería unos pocos bits de información en grandes cantidades.

Además, los parámetros de actualización de la rampa es una operación que va a pasar con regularidad (véase la trayectoria de un módulo para entender eso). Debemos asegurarnos de que el cambio de los parámetros de la función de rampa en cualquier momento.

Prueba

He aquí un resumen de la salida del filtro. La curva roja corresponde a la entrada y la curva de color verde corresponde a la salida. A veces, las características de los filtros se cambian de forma dinámica:

en t = 0

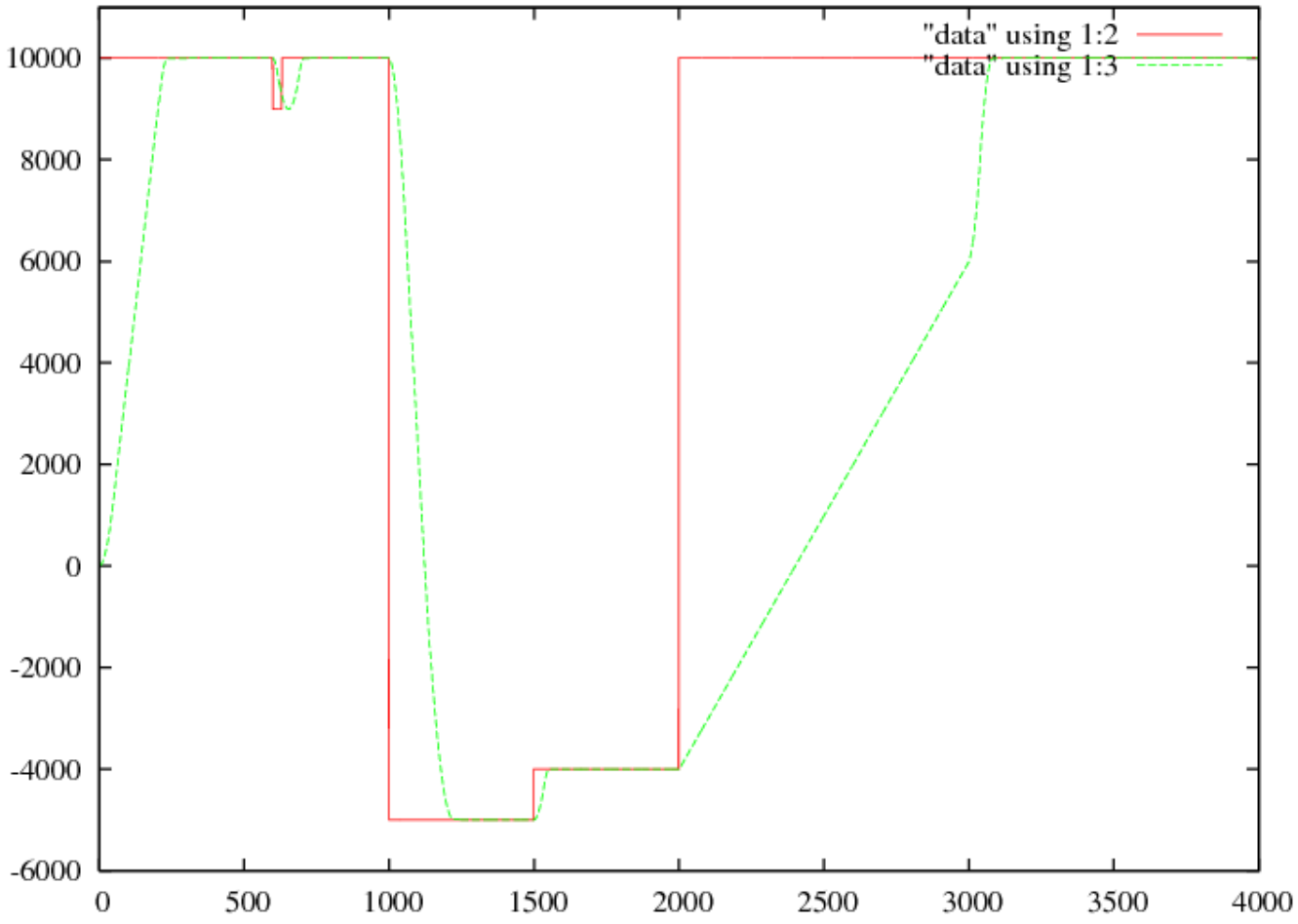
```
quadramp_set_1st_order_vars (& q, 50, 100); quadramp_set_2nd_order_vars (& q, 1, 2);
```

en t = 2000:

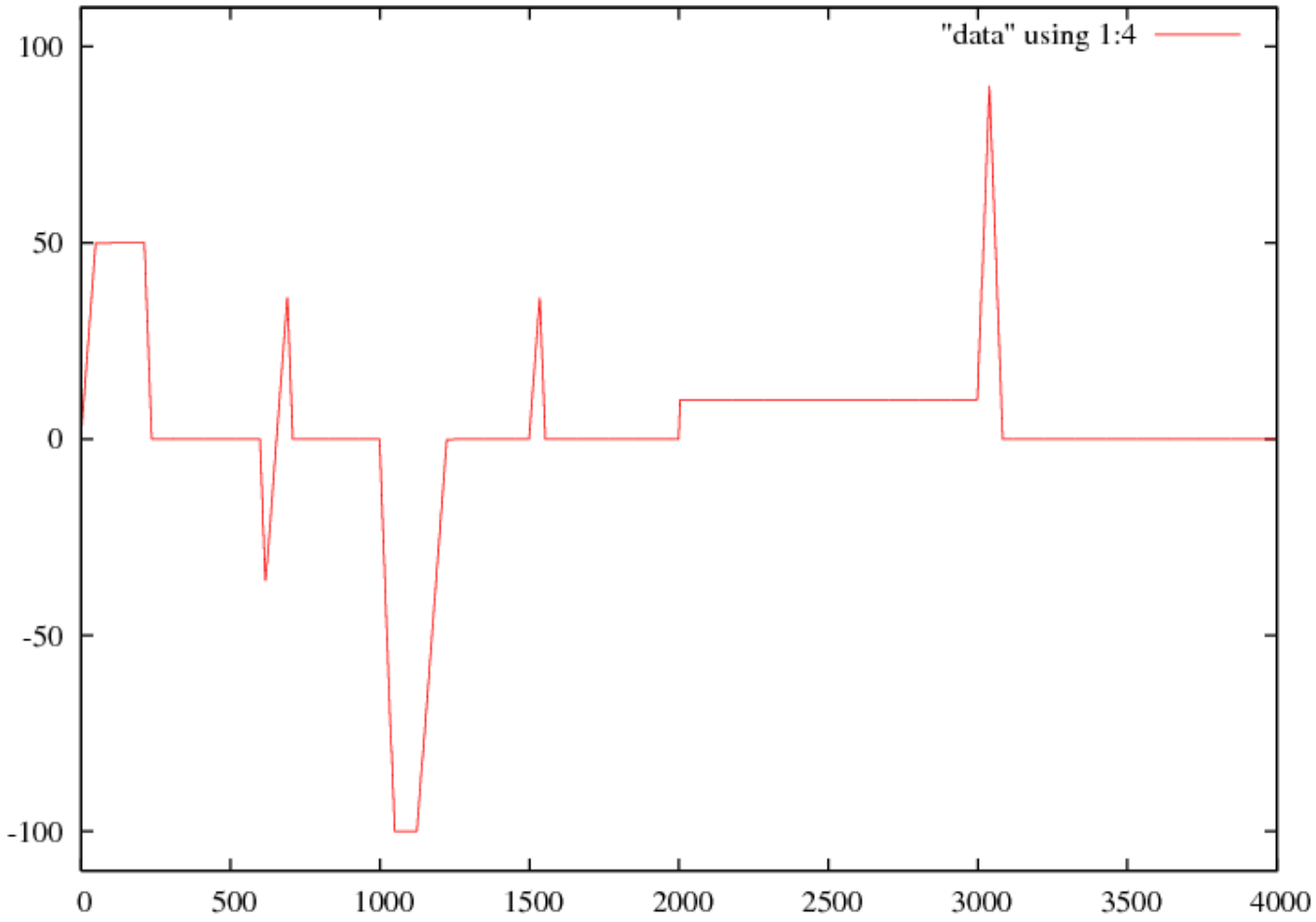
```
quadramp_set_1st_order_vars (& q, 10, 10); quadramp_set_2nd_order_vars (& q, 2, 2);
```

en t = 3000:

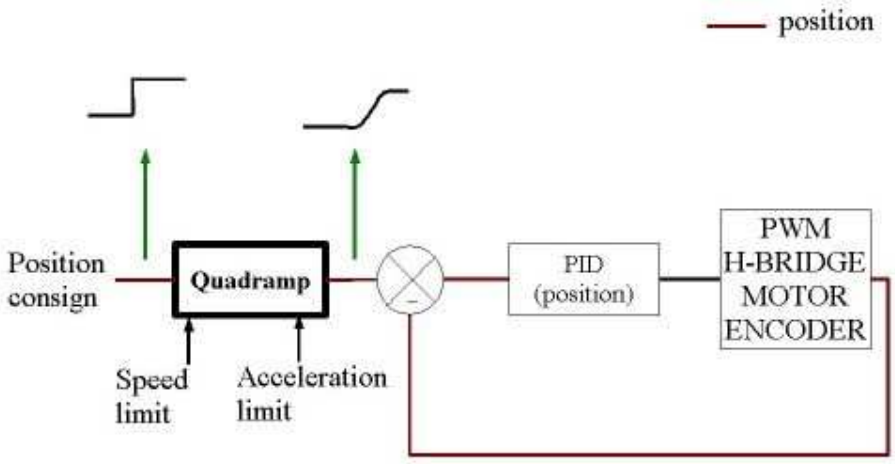
```
quadramp_set_1st_order_vars (& q, 100, 100);
```



La segunda curva muestra el mismo ejemplo y muestra que la curva de velocidad es trapezoidal (donde la velocidad máxima se alcanza, de lo contrario, es triangular).



Aquí hay un diagrama de la sujeción completa:



Programa de ejemplo

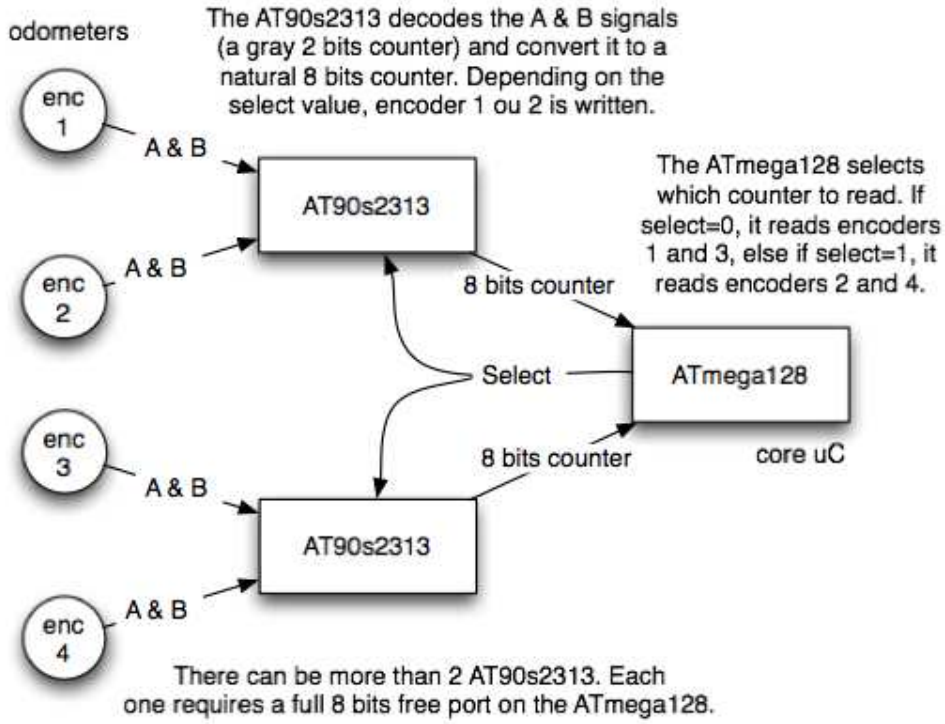
```
estructura cs cs; estructura p pid_filter, estructura qr quadramp_filter; # define 10000/SCHEDULER_UNIT PERIO
```

Interfaz con el PWM y cuentakilómetros

En todos los ejemplos anteriores, hemos supuesto que las funciones de control de los motores y leer los valores del cuentakilómetros se proporcionaron. Sin embargo, la generación de la señal PWM requiere la configuración de los registros internos en abril, y leyendo acerca de los codificadores que requiere un hardware específico para descifrar las señales.

dispositivos / codificadores / encoders_microb

En la actualidad, hay en aversivo dos módulos para la lectura de odómetros ajustado al 2 de hardware diferentes. La de Eirbot y la de Microb Tecnología vamos a detallar. Desde el punto de vista electrónico, y dada la Atmega128, la operación es relativamente simple: el valor de los codificadores llega en un bus de 8 bits compartida por dos codificadores. El codificador es el codificador seleccionado par o impar según el estado del pin de selección.



Interfaz

```
vacío encoders_microb_init (void);
```

Encoders_microb inicialización del módulo. Esta función ajusta a 0 los contadores internos al módulo.

```
vacío encoders_microb_manage (void);
```

Esta función lee el estado del autobús para recuperar el valor de codificadores con un número par, entonces cambia el estado del pin de selección para preparar la siguiente lectura del codificador impar. La llamada siguiente a la inversa. Esta función se debe llamar de vez en cuando entre las dos lecturas del mismo codificador, su valor no ha cambiado más de 127. En este caso, el robot de 2006, esta función se llama cada 128us, para la lectura de un solo codificador cada 256us.

```
encoders_microb_get int32_t (void * encoders_microb_data);
```

encoders_microb_get () para leer el medidor bruto. Encoders_microb_data argumento es un puntero que se convierte a (int) y representa el número de codificador. La ventaja de un void * que el prototipo está siendo compatible con otras implementaciones que requieren la aprobación de una estructura completa.

de hardware / PWM

Descripción

El módulo de PWM proporciona una función para su producción con el servo, es decir, la función void `process_in (void *, int32_t)` . Se puede generar una señal en un pin PWM (Pulse Width Modulation), es decir, una onda cuadrada periódica cuyo ciclo de trabajo varía. Es esta señal (además del bit de signo), que será amplificado por LM18200T y se utiliza para controlar los motores. En este módulo se pueden generar dos señales en un comando.

Interfaz

- `vacío pwm_init (void);`

PWM de inicialización del módulo.

- `vacío pwm_set (void *, int32_t);`

La interfaz mínima para trabajar con la esclavitud es una función que el anterior. El parámetro utilizado por primera vez el miedo a pasar por ejemplo, el número de PWM.

Ejemplo

Podemos tomar el ejemplo de la sección anterior, agregando que la gestión de PWM y codificadores.

```
estructura cs cs; estructura p pid_filter, estructura qr quadramp_filter; # define 10000/SCHEDULER_UNIT PERIO
```

Esclavitud polares (ángulo / distancia)

Ahora nos centramos en una aplicación más específica a un robot para cortar tratando de esclavizar a su ángulo y la distancia con dos motores, mientras que la identificación de la materia. Vamos a volver a utilizar para todo lo que hemos visto anteriormente.

dispositivos / robot / robot_system

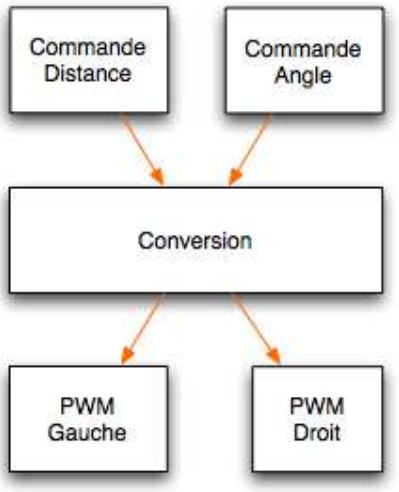
Descripción

Hemos visto más arriba, no queremos esclavizar a las ruedas de forma independiente el uno del otro. Hemos optado por esclavizar a dos tamaños que dependen de:

- la suma de los valores devueltos por los dos codificadores de las dos ruedas dividido por 2 (distancia = theta)
- A diferencia de los valores devueltos por los dos codificadores de las dos ruedas dividido por 2 (ángulo alfa =)

En este módulo se realiza esta operación simple para los codificadores de la lectura (`angle_get ()` y `distance_get ()`) y para el envío de PWM (`angle_set ()` y `distance_set ()`).

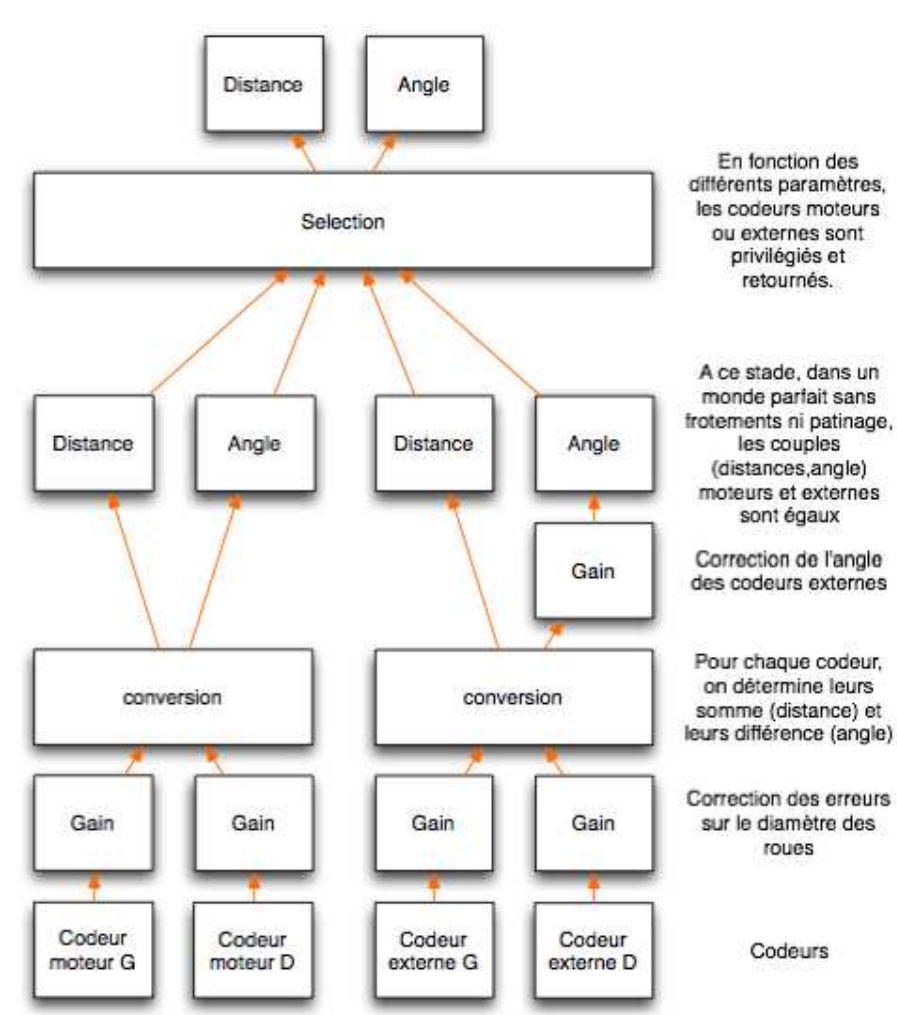
El ángulo de PWM y la distancia, por supuesto, no existen físicamente. Su interfaz es el mismo que el PWM real, que puede ser utilizado como una función `process_in` en el administrador de sistema de control.



Para los codificadores incrementales, el trabajo es más complejo porque el objetivo es aprovechar los dos pares de

codificadores (motores y externo). Hemos visto, tenemos nuestros codificadores robot en cada uno de los motores, sino también una rueda loca (sin motor). Para impulsar el robot, así que tenemos odómetros gestión 4. intentos Robot_system módulo de utilizar la información de los 4 codificadores para codificadores ofrecen ángulo virtual ya distancia.

La idea es utilizar los motores de programadores independientes o codificadores externos, y puede cambiar de uno a otro en cualquier momento. Se trata de algo de matemáticas para que los codificadores devolver la misma cosa, a pesar de que se colocan en diferentes lugares. Una última característica es que el módulo le permite aplicar coeficientes correctores a los codificadores para compensar los defectos mecánicos (pequeño error en el diámetro de la rueda, por ejemplo). De esta manera, sólo queda para multiplicar la esquina para ir a los motores fuera de codificadores codificadores.



Estructuras

- angle_distance.h en:

```
struct {rs_wheels int32_t izquierda, derecha int32_t;}
```

Estructura que describe las coordenadas polares del robot.

```
struct {int32_t ángulo rs_polar int32_t distancia;}
```

Estructura que describe los contadores de las ruedas.

- en robot_system.h

```
struct {robot_system banderas uint8_t; estructura pmot_prev rs_polar; estructura pext_prev rs_polar; estructu
```

Esta estructura de las tiendas de toda la información necesaria para el buen funcionamiento de robot_system. Los punteros a las funciones de entrada / salida (PWM y codificadores), las ganancias que se aplicará a cada uno de los codificadores, la relación de los codificadores de canal canales y motores, contadores internos y las banderas.

Interfaz angle_distance.h

```
vacío rs_get_polar_from_wheels (struct * p_dst rs_polar, estructura rs_wheels w_src *);
```

La conversión de una estructura que contiene los valores de los codificadores de una estructura que contiene la suma de dos programadores, y su diferencia de 2.

```
vacío rs_get_wheels_from_polar (struct * rs_wheels w_dst, estructura * p_src rs_polar);
```

Revertir la conversión de la función anterior.

Interfaz robot_system.h

```
vacío rs_init (struct RS robot_system *)
```

Inicializa la estructura de RS: el valor 0 de toda la estructura, excepto relación a 1,0

```
vacío rs_set_ratio (struct RS robot_system *, relación de dos personas);
```

Define la relación entre la distancia entre las ruedas de los motores y la distancia entre las ruedas exteriores.

```
vacío rs_set_left_pwm (struct robot_system * RS, void (* left_pwm) (void *, int32_t),
left_pwm_param void *);

vacío rs_set_right_pwm (struct robot_system * RS, void (* right_pwm) (void *, int32_t),
right_pwm_param void *);

vacío rs_set_left_mot_encoder (struct RS robot_system *, int32_t (* left_mot_encoder) (void *),
left_mot_encoder_param void *, aumento doble);

vacío rs_set_right_mot_encoder (struct RS robot_system *, int32_t (* right_mot_encoder) (void
*), right_mot_encoder_param void *, aumento doble);

vacío rs_set_left_ext_encoder (struct RS robot_system *, int32_t (* left_ext_encoder) (void *),
left_ext_encoder_param void *, aumento doble);

vacío rs_set_right_ext_encoder (struct RS robot_system *, int32_t (* right_ext_encoder) (void
*), right_ext_encoder_param void *, aumento doble);
```

Especifica las funciones de entrada / salida: a la izquierda PWM y codificadores de derecha e izquierda y derecha. Para los codificadores, también especifica la ganancia utiliza para corregir el valor de cada codificador.

```
vacío rs_set_angle (RS void *, el ángulo int32_t)
```

actualizaciones de PWM para reflejar el nuevo punto de vista de control, y last_command copia de seguridad.

```
vacío rs_set_distance (RS void *, dist int32_t)
```

actualizaciones de PWM para reflejar el nuevo control a distancia, y last_command copia de seguridad.

```
rs_get_angle int32_t (RS void *)
```

Devuelve el ángulo actual

```
rs_get_distance int32_t (RS void *)
```

Devuelve la distancia actual

rs_get_ext_angle int32_t (RS void *)

rs_get_mot_angle int32_t (RS void *)

rs_get_ext_distance int32_t (RS void *)

rs_get_mot_distance int32_t (RS void *)

Lo mismo, pero preguntando específicamente los codificadores distancia o el ángulo como motores o rodillos.

rs_update void (RS void *);

Lee los codificadores, y realiza los cálculos para actualizar los codificadores virtual interna.

rs_set_flags vacío (struct RS robot_system *, banderas uint8_t);

Por ahora la única bandera es RS_USE_EXT utilizados para determinar si se utiliza codificadores externos o motores de codificadores para la conversión al ángulo y la distancia. Esta función puede ser llamada en uso sin interrumpir la operación.

Cálculo de la posición (x, y, a)

math/vect2

Este módulo proporciona funciones y estructuras de datos para su uso en vectores 2D cartesiano o polar. Vectores mediante el escalar *real* de su información de contacto (por defecto es un *doble*). Está documentado aquí .

dispositivos / robot / position_manager

Descripción

El módulo de funciones position_manager recuperación de la posición del robot. Se trata de un "administrador", ya que proporciona una position_manage function () cuya función es actualizar la posición del robot en el campo. Por lo tanto, mantiene la actualización de la posición del robot en el campo (x, y, a) la utilización robot_system.

Estructuras

```
struct {track_cm robot_physical_params doble; distance_imp_per_cm doble;}
```

Estructura que describe el número de pulsos a un centímetro, anís que los codificadores manera real (o virtual) del robot.

```
struct {xya_position doble x, y doble, un doble;}
```

Estructura para almacenar una posición cartesiana del robot en el suelo. El ángulo en radianes se almacena.

```
struct {xya_position_int16_t int16_t x, y int16_t, int16_t uno;}
```

La misma estructura, pero los valores son números enteros. El ángulo se almacena en grados.

```
struct {struct robot_physical_params robot_position phys; estructura xya_position pos_d; estructura pos_int16
```

Estructura de almacenamiento de la posición instantánea del robot, y un puntero a la robot_system estructura que se describe cómo recuperar el valor de codificadores de ángulo y la distancia.

Interfaz

```
vacío posistion_init (struct robot_position * pos)
```

Inicializa la estructura de ubicación (sólo 0).

```
vacío posistion_set_related_robot_system (struct pos robot_position *, struct * robot_system
RS)
```

Copia de seguridad en el puntero pos de la estructura asociada robot_system. estructura de R se utiliza para recuperar el valor del ángulo de codificadores virtual ya distancia.

```
vacío position_set (struct pos robot_position *, int16_t x, y int16_t, int16_t a);
```

Establece la posición actual del robot.

```
position_set_physical_params vacío (struct pos robot_position *, track_cm doble,
distance_imp_per_cm doble);
```

Define los parámetros físicos del robot (distancia entre las ruedas, el número de pulsos por pulgada.

```
vacío position_manage (struct robot_position * pos)
```

Vuelve a calcular la posición absoluta (x, y, a) de acuerdo a la posición del delta de las ruedas y los parámetros físicos del robot.

```
int16_t línea position_get_x_s16 (struct robot_position * pos);

int16_t línea position_get_y_s16 (struct robot_position * pos);

int16_t línea position_get_a_deg_s16 (struct robot_position * pos);

position_get_x_double en línea doble (estructura robot_position * pos);

position_get_y_double en línea doble (estructura robot_position * pos);

position_get_a_rad_double en línea doble (estructura robot_position * pos);
```

De acceso.

Detección de interbloqueos

robot_control / blocking_detection_manager

Descripción

La detección de interbloqueos es útil en el caso de controlar un robot para cortar E = M6. Se identifican los obstáculos y por lo tanto actuar en consecuencia mediante la realización de este giro a su alrededor.

Hay por lo menos dos tipos de obstáculos:

- 1 - La unidad de las ruedas giran. En este caso, se puede detectar el punto muerto al señalar que los valores devueltos por los codificadores de los motores y los codificadores de la rueda son incompatibles.
- 2 - Las ruedas se bloquean. El error del servo aumenta hasta que alcanza un umbral crítico. Tener un servo que realiza rampas de velocidad simplifica este tipo de detección de punto muerto, porque consideramos que los parámetros de la rampa se ajustan de manera que el motor esté físicamente en condiciones de aplicar esta rampa.

En la práctica, ya que este año es probable que no tenía codificadores de ruedas motorizada, se utiliza un único algoritmo para detectar bloqueos (pero probado). El módulo se control_system recuperar la información del módulo de manera

independiente. La esclavitud "control" debe ser un control de la rotación se ha seleccionado. Esto nos permite utilizar este módulo en tanto las ruedas del robot en otro actuador motorizado. Lo más difícil es establecer los umbrales para encontrar un buen compromiso entre una reacción exagerada o no lo suficientemente rápido.

La siguiente función se denomina intervalos regulares para detectar bloqueos.

```
Si la velocidad medida excede un umbral y que el error de control es mayor que otro umbral Si cpt <CPT_MAX cp
```

Otra característica poco en cualquier momento se llamó para preguntar si el bloqueo:

```
Si cpt == 1 return 0 etourner CPT_MAX
```

Interfaz

vacío bd_init (struct * blocking_detection bd, estructura CS CS *)

Módulo de inicialización

bd_set_thresholds vacío (struct * blocking_detection bd, speed_thres uint32_t, err_thres uint32_t, cpt_thres uint16_t)

Inicializa los umbrales

vacío bd_reset (struct * blocking_detection bd)

Restablece el contador a 0. Esta función debe ser llamada cuando un bloqueo se ha tenido en cuenta.

vacío bd_manage (struct * blocking_detection bd)

Una llamada a actualizar periódicamente los bloqueos de la información.

bd_get uint8_t (struct * blocking_detection bd)

Devuelve 1 en caso de empate.

Trayectoria generación

dispositivos / robot / trajectory_manager

Descripción

Estamos entrando en una parte difícil. Este módulo se utiliza para definir las directrices para la esclavitud de manera que pueda controlar el robot en forma sencilla, por ejemplo, especificando un destino (x, y). La función básica consiste en agregar un evento periódico (aproximadamente cada 0,1 segundos) que actualizar las directrices para la distancia y el ángulo de la esclavitud. El módulo también proporciona funciones simples que permiten la interconexión entre el servo (pasos de trabajo) y el promotor / estratega (que se siente más cómodo con centímetros).

Interfaz

Aún en desarrollo, pero usted puede conseguir una buena idea de la versión final aquí (http://cvsweb.droids-corp.org/cgi-bin/viewcvs.cgi/aversive/modules/devices/robot/trajectory_manager/trajectory_manager.h?revision=1.4.4.5&view=markup&pathrev=b_zer0)

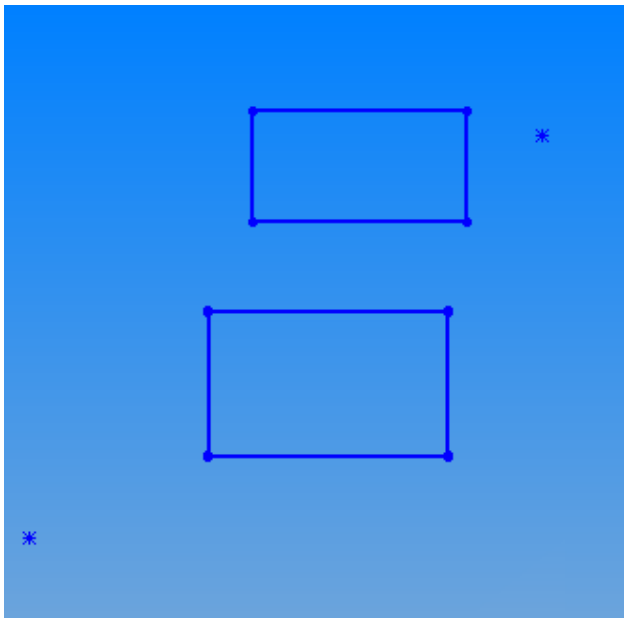
dispositivos / robot / obstacle_avoidance

Descripción

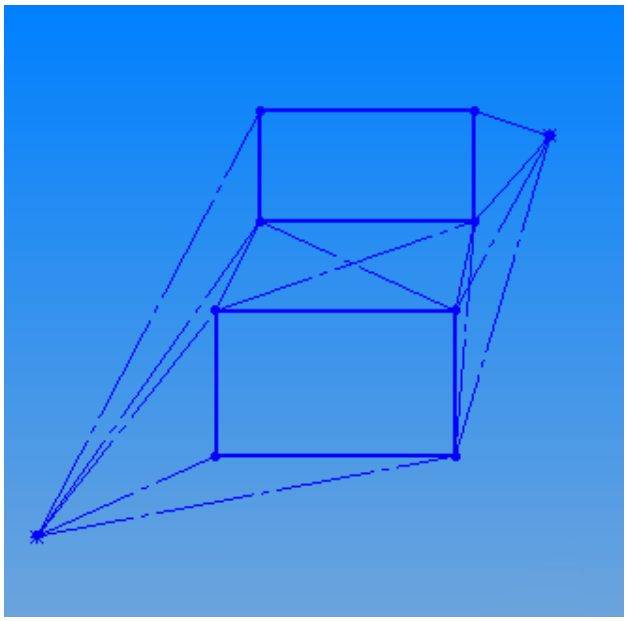
Este módulo puede generar trayectorias con un grado de alrededor de una lista de obstáculos.

El algoritmo utilizado es el algoritmo de "punto visible (tal vez no sea el nombre real). Se toma como entrada una lista de polígonos (convexos, para abreviar), que representa los objetos para evitar el cálculo de la trayectoria. El punto de partida y llegada, así como las dimensiones del área de juego también se proporcionan.

Con estos datos, el algoritmo trata de encontrar el camino más corto que conecta el punto de partida hasta el destino y evitar objetos en el patio En el siguiente ejemplo, la mano del robot lugar por la izquierda y debe llegar al punto en la parte superior derecha evitando los dos rectángulos (dos robots de oposición, por ejemplo).



Inicialmente, el algoritmo atraeré a todos los segmentos de conectar todos los puntos con el fin de "dirigir" el uno del otro, lo que significa que todos los puntos de "ver". En nuestro ejemplo, se trata de líneas de puntos.

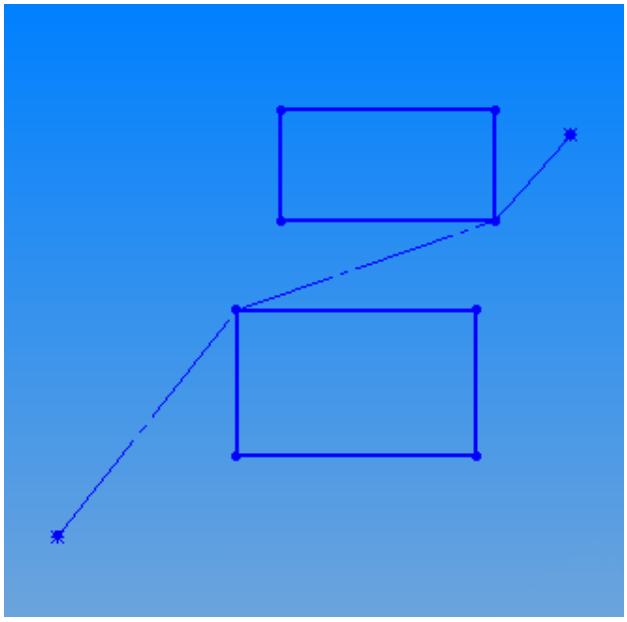


Cabe señalar que las líneas que forman los lados de los polígonos se generan (si los polígonos no se intersectan). Obtenemos un gráfico cuyas aristas son los segmentos generados anteriormente. El objetivo es entonces la ponderación de cada segmento por su longitud. Así, un segmento de longitud 2 tiene un peso de 2, y así sucesivamente.

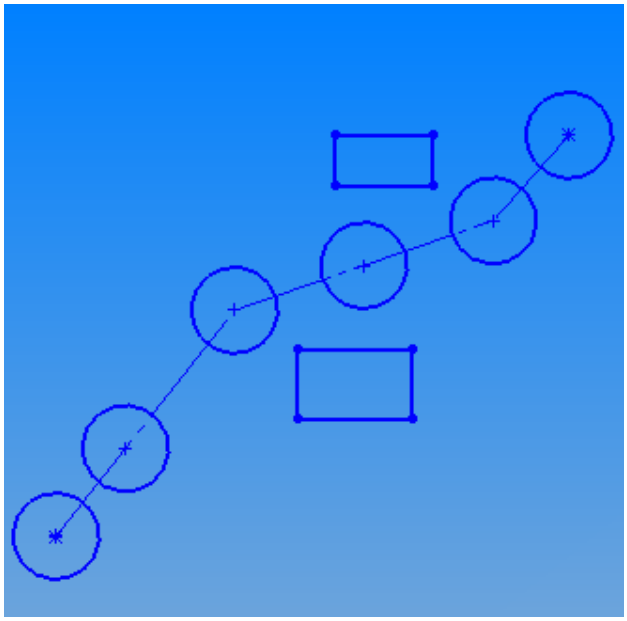
Llegamos a la conclusión calculando la ruta más corta (en términos de peso) que conecta el punto de partida hasta el destino, esto se hace usando el algoritmo de Dijkstra (http://fr.wikipedia.org/wiki/Algorithme_de_Dijkstra)

Resultados

El robot era entonces sólo tienes que seguir el siguiente segmento devueltos.

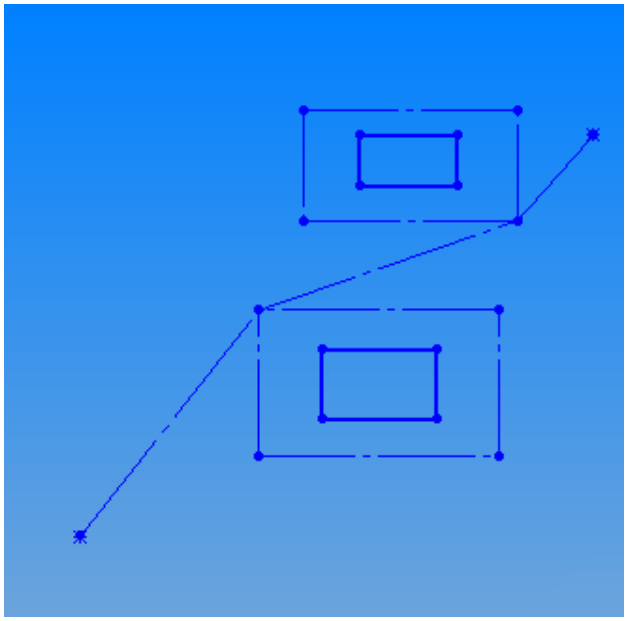


Aquí está el resultado final (el robot está representado por un círculo). El robot alcanza el punto final, evitando obstáculos.

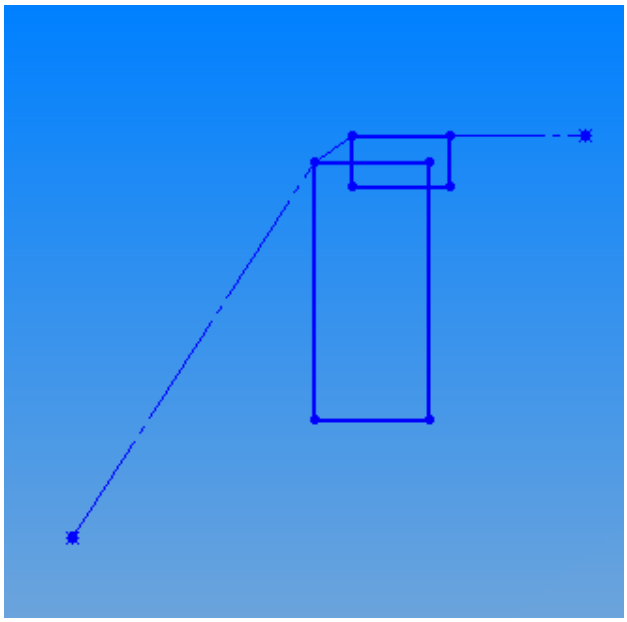


Tenga en cuenta que este algoritmo da el camino más corto para una configuración determinada en el cálculo de la trayectoria. Si los elementos para evitar no son fijos, otros cálculos de trayectoria sin duda necesario.

Otro punto importante es que el camino volvió roza las copas de los polígonos, lo que significa que, en realidad, nuestro robot sólo se pierda el otro robot u objetos en el campo. Para superar este problema, una solución consiste en exagerar el tamaño de los objetos de evitar. Aquí los objetos reales (líneas continuas) estuvieron representados por polígonos un poco más grande.



El algoritmo también pasa a calcular la ruta correcta, incluso si los polígonos se intersectan. En efecto, si un vértice está incluido en otro polígono, no hay segmento estará vinculado (la cumbre no es "visto" por los otros vértices)



Video

En la imagen: direct.3gp Trajkovski , tenemos una primera trayectoria sin obstáculos a través de los 4 puntos en el campo, los pocos primeros centímetros delante de él, el segundo lo suficiente a su izquierda, luego se da vuelta y regresa a terceros, y encadenado a este último punto en la parte inferior izquierda en el video.

En el segundo video la imagen: avoid.3gp Trajkovski , ponemos un obstáculo en el centro del campo. Los más información solo está disponible para la posición del robot y el tamaño del obstáculo. Preguntas para realizar el camino de nuevo, pero algunas partes sería imposible de hacer sin tocar el obstáculo. A continuación, agrega automáticamente nuevos puntos de cruce.

Conclusión

Esta documentación describe el funcionamiento del servo Microb Tecnología para el año 2008. Como se señaló anteriormente en esta página, que aún está en evolución y muchas cuestiones aún puede ser incompleta o inconsistente, incluso. Estoy abierto a cualquier comentario.

Hemos visto que el control de un robot para cortar no es una cosa fácil, pero separar el programa real de varios ladrillos que se pueden montar, es más fácil de ver con claridad. Este documento puede servir de base para alguien a quien le gustaría

volver a implementar un control de Eurobot, o la documentación de nuestra esclavitud. Al igual que muchos equipos lo han hecho, es muy posible recuperar una parte del código y adaptarlo en un robot diferente del nuestro.