Ceres – Código principal – Arduino Mega

Por: Ing. Guillermo Sánchez Herrera.

Objetivo: Explicar en detalle el funcionamiento del código principal que permite la locomoción del robot Ceres, el cual esta embebido en un Arduino Mega y se comunica por medio de ROS.

Nota: Este documento es una guía anexa al Manual de operación Ceres – Agrobot, para detalles de la interconexión del módulo del Arduino principal a los demás elementos del robot remítase a esa guía.

Recursos: El código y las librerías necesarias para su ejecución están disponibles en la carpeta Drive del proyecto de 2019:

> INV-ING-2990 - Self driving Ceres 2019 > Propuesta y paquetes de trabajo_Seguimiento

> ROS > CODIGO- NO MODIFICAR

En la carpeta CeresMainArduino_2020.

Dentro de esta se encuentran dos .rar con los códigos mencionados en el presente documento.

1 Descripción del modulo

El modulo Arduino mega principal del robot, permite la generación de los voltajes análogos a los motores principales del robot. En la figura 1 se observa el Arduino mega y el shield realizado por el tesista Adrien Legrand que permite la conexión de los ADC y las señales de los motores (parte superior derecha).

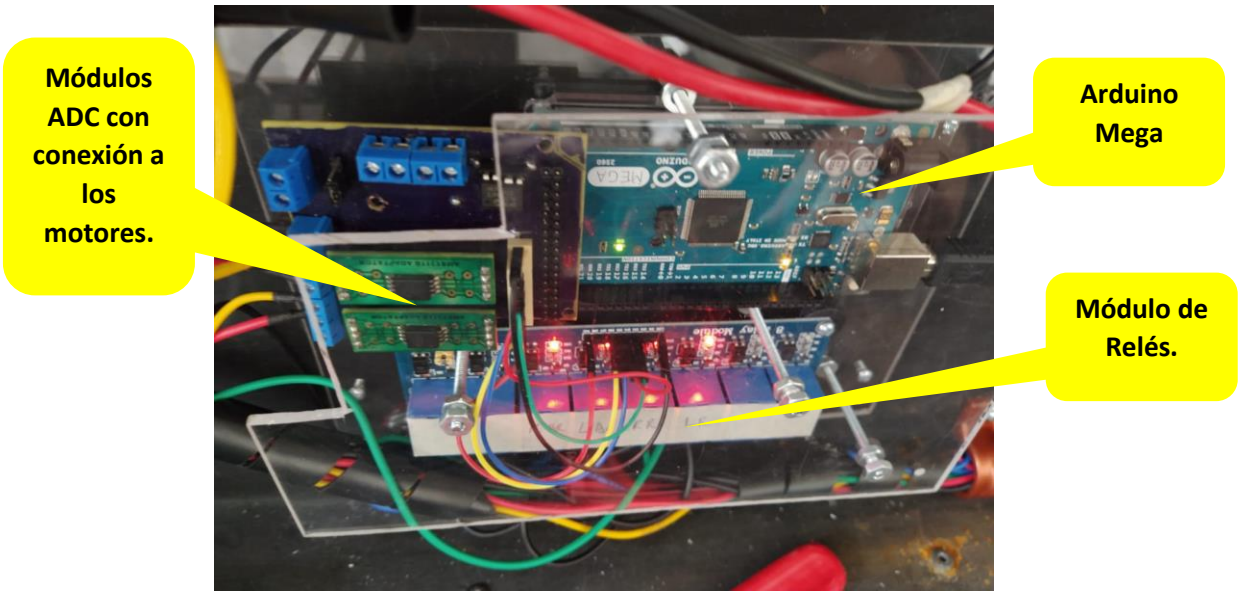

Figura 1: Modulo de control de locomoción del robot Ceres

Interconexiones físicas: El modulo fijado a la pared posterior del robot se conecta con:

- El computador principal por medio del cable USB de Arduino.
- Las salidas del shield son conectadas a las señales de Vcc, Ground y Analog de los dos motores (6 cables).
- Se conectan 4 señales lógicas más alimentación y tierra al módulo de relés que permite la ejecución de comandos de: Frenado y Reversa para cada uno de los motores. Estos relés se encuentran debidamente marcados en la parte superior, para facilitar cualquier revisión.

Interconexiones de software:

Entradas:

- Radio control: El programa monitorea la existencia del topic **/ceres/RC**, en especial el canal de la parada de emergencia.
- IMU: Debido al ancho de banda ocupado en la transmisión de todas las mediciones de la IMU, se hizo necesario la creación de un topic llamado **/advanced_navigation_driver/twist2D** el cual contiene solo la velocidad angular y lineal medida por la IMU, la cual es usada para la operación del robot en lazo cerrado.
- Referencias en velocidad angular y lineal: El modulo recibe estas referencias desde ROS por medio del topic **/ceres/cmd_vel**

Salidas:

- Como salida el módulo publica el topic **/ceres/controlsignal** con el fin de poder verificar hasta 4 variables de la operación correcta del módulo.

Nota: Este último topic fue editado de la versión original con el fin de dar compatibilidad al código con los mensajes preestablecidos de ROS en pro de integraciones con Matlab.

2 Modos de operación

El programa cuenta con tres modos o estados principales, estos pueden ser seleccionados por el operador con el radio control:

- **Modo Manual:** El programa toma las señales del radio control como referencias para la velocidad lineal y angular del robot.
- **Modo Automático:** El programa toma como referencias los datos publicados en el topic **/ceres/cmd_vel**, el cual puede ser publicado desde un script que lea un archivo o un control de alto nivel.
- **Modo Emergencia:** El programa detiene su ejecución, envía el valor mínimo de voltaje a los motores e indica por medio de una alerta de emergencia a ROS del estado en parada.

2.1 Lazo de control de bajo nivel

Con el fin de evitar malos funcionamientos y debido a la necesidad de **calibrar la IMU en caso de dejar de operar el robot por más de 48 horas**: El cierre del lazo de control interno se encuentra normalmente deshabilitado, lo que implica que los voltajes enviados a los motores dependen de la señal de referencia que se envié por el radio control y una ganancia en lazo abierto predefinida.

Si se desean realizar pruebas con el lazo de control de bajo nivel activo, se debe sacar el robot del parqueadero en modo manual, calibrar la IMU y luego cambiar la bandera **closeLoop** al estado **True** la cual se encuentra en la **línea 46** del código.

```
#define closeLoop false
```

3 Código principal

El código principal se encuentra documentado por Adrien Legrand, en esta sección se detallaran aspectos que permitan una mejor comprensión del código.

Nota: Se cuenta con dos códigos en la carpeta referenciada previamente, CeresMainV1_PIDDiscretoF, CeresMainArduino_2019_V1, estos dos códigos tienen implementaciones diferentes del control PID, sin embargo, funcionan con el robot ceres, para el caso de CeresMain se realizo una sintonía con prueba y error del PID. En el caso del PID discreto, se realizo una verificación mediante HIL de su correcta implementación como PID discreto, sin embargo, luego de realizar sintonización en simulación el desempeño no fue el deseado, debido a aspectos no modelados.

Se cuentan con siete secciones principales del código las cuales se enumeran a continuación y se detallan en la siguientes sub secciones:

I - Libraries Including

II- Parameters Definition

III- Headers Definition

IV- ROS Nodes Definition

V- Global Variables Definition

VI- Loop

VII- Functions Declaration

3.1 Librerías

El código realiza una verificación de la versión de la tarjeta a programar e incluye las librerías de ROS (ros.h), los tipos de mensajes que son usados en el código y las librerías SPI y Timer.

Nota: Estas librerías deben estar en la carpeta de proyectos de Arduino, la cual está por defecto en: C:\Users\Name\Documents\Arduino la ubicación exacta en su computador se puede verificar en Archivo – preferencias.

Dentro de esa carpeta debe existir una subcarpeta llamada **libraries** donde se descompriman las librerías ubicadas en la carpeta drive.

3.2 Definición de parámetros

En esta sección se definen parámetros básicos de operación y cálculos a realizar en el programa, siendo los principales:

#define closeLoop false

#define autoTimeout 500 // Timeout for ROS Input

#define RCTimeout 1000 // Timeout for RC input

#define IMUTimeout 50 // Timeout for IMU Input

Nota: Estos Timeouts en milisegundos son importantes para la verificación de la recepción de los datos de entrada, en caso de no cumplirse estos tiempos el programa pasara al modo de parada de emergencia y avisara el error.

#define ControlTime 20 //time for control interrupt Linea 62, define el tiempo de operación del control de bajo nivel 20ms en este caso.

Seguido de las ganancias de los controladores PID: Pu, Iu, Du para velocidad lineal y Pw. Iw y Dw para la velocidad angular.

Finalmente se definen los parámetros de velocidades máximas y velocidades máximas en modo manual con los parámetros:

```
#define minVel 0.0    // Maximum Robot Linear Velocity [m/s]
#define minAngVel -0.25 // Maximum Robot Angular Velocity [rad/s]

#define maxVel 1.0    // Maximum Robot Linear Velocity [m/s]
#define maxAngVel 0.25 // Maximum Robot Angular Velocity [rad/s]

#define maxManualVel 1.0    // Maximum Robot Linear Velocity in Manual Mode [m/s]
#define maxManualAngVel 0.25 // Maximum Robot Angular Velocity in Manual Mode [rad/s]
```

3.3 Definición de encabezados

En esta sección se nombran las funciones a ser programadas, en este caso se cuenta con:

Tres funciones para la recepción de información del radio control, Imu y referencias.

Dos funciones para el PID de velocidad angular y lineal.

Por último, la función que realiza el calculo final de velocidad a aplicar, convierte a voltios y envía el comando a los DAC (applySpeed).

3.4 Definición de los nodos de ROS

En esta sección se inicializa lo referente a ROS, configurando los tres suscriptores para los datos de Radio control, IMU y referencia previamente descritos. También se define el mensaje log_msg2 a ser llenado con las variables a monitorear por medio del topic /ceres/controlSignal.

Se dejan como comentario la implementación anterior del topic de monitoreo correspondientes al topic /ceres/ardulog, la cual se cambio por compatibilidad como se detalló anteriormente.

3.5 Definición de las variables globales

Acá se definen todas las variables a ser usadas durante la operación del código, siendo la más importante la variable del modo de operación la cual es inicializada en emergencia.

```
int mode=2; // 0: RC/Manual, 1: Automatic, 2: Emergency Stop
```

Hacia el final de esta sección se realiza el llamado a la función setup() de Arduino donde se realizan las siguientes ordenes:

- Se inicializan los DAC
- Se inicializan los nodos de ros
- Se inicializa el timer que ejecutara el controlador por medio de la función applycontrol
- Se asignan los pines de parada de emergencia, freno y reversa.
- Se inicializan las salidas de los pines. Esta inicialización no debe cambiarse pues permite el correcto desplazamiento del robot teniendo en cuenta el cableado actual del mismo.

3.6 Loop

En este bucle principal se realiza el llamado de las funciones y el manejo de variables que permite la locomoción del robot Ceres.

En la primera parte se realiza la detección del cambio de modo de operación y se informa a ROS de ese cambio de modo, también se activan pines de la tarjeta para el sentido de giro (este realiza una validación: ((revCH < -60) && !closeLoop), esto permite realizar el cambio del sentido de giro de los motores, si se coloca la palanca de velocidad angular a la izquierda y se cambia de modo, esto con el fin de parquear o sacar el vehículo del laboratorio) y la parada de emergencia (este último no está conectado a ningún hardware pero se puede usar en futuras aplicaciones), recordando que este modo depende de la variable **mode** y es asignado dentro de la función **receiveRC**, ver las siguientes secciones para más detalles.

La siguiente y última sección es un switch-case que depende del modo actual de operación, este con una serie de if encadenados que dependen de variables monitoreadas, a continuación, se realiza una breve descripción de lo que se realiza para cada modo:

- **Modo 0 – Manual**

En este caso se busca que las señales de radio control sean tomadas como referencias y enviadas a los motores.

Se inicia con una validación de la recepción de datos del radio control, que permite identificar la desconexión o mal funcionamiento de este. En el caso de no cumplirse este requisito, se envía una velocidad de 0.0 a la función **applySpeed** para detener los motores y se notifica a ros del vencimiento del Timeout.

En el segundo if se realiza la validación de la bandera **closeLoop,** la cual permite cerrar el lazo para tomar en cuenta las mediciones de la IMU y ejecutar el PID.

Si closeloop es True:

Se realiza una última validación para el tiempo de recepción de datos de la IMU, en caso de no cumplirse al igual que con el tiempo de recepción del radio control, se detienen los motores y se notifica a ROS.

Si se cumple la recepción de datos de la IMU, se realiza el llamado a la función **tcontrol.update()** la cual notifica al timer que se desea seguir ejecutando la función **applycontrol**, esta definición se

realiza en el **Setup** línea 170. Los detalles de la función **applycontrol** se especifican en secciones posteriores.

Si closeloop es false:

Se asignan las variables u y w para las velocidades lineal y angular escalando el valor máximo configurado (ver sección Parámetros) con el rango del radio control (0 a 100).

Finalmente se aplica las velocidades asignadas por medio de la función **applySpeed**.

Nota: En esta sección hay un código comentado que permite verificar esas velocidades u otros datos que se requieran por el topic publicado /ceres/ardulog, se conserva esta sección a modo de ejemplo.

- **Modo 1 – Automático**

En este modo se tomarán las referencias del topic /ceres/cmd_vel y se aplicaran a los motores.

Debido a que el principio de funcionamiento del modo depende del topic **/ceres/cmd_vel**, se realiza la verificación de recepción de la referencia. Si no se cumple un tiempo máximo de 500ms entre referencias, se detienen los motores y se notifica por a ROS.

De manera similar al modo manual, se realiza una validación de la **bandera de closeloop**, esta funcionalidad se conserva, debido a que puede ser del interés de los investigadores probar una serie de referencias en lazo abierto con propósitos de identificación. A continuación, se detalla el procedimiento en los dos casos:

Si closeloop es True:

Se realiza una última validación para el tiempo de recepción de datos de la IMU, en caso de no cumplirse al igual que con el tiempo de recepción de las referencias, se detienen los motores y se notifica a ROS.

Si se cumple la recepción de datos de la IMU, se realiza el llamado a la función **tcontrol.update()** la cual notifica al timer que se desea seguir ejecutando la función **applycontrol**, esta definición se realiza en el **Setup** línea 170. Los detalles de la función **applycontrol** se especifican en secciones posteriores.

Si closeloop es false:

Se definen y asignan las variables u y w para las velocidades lineal y angular tomando el dato recibido por el topic de ROS.

Si se tiene configurada a True la variable de **closeloop** se realiza el llamado a las funciones **PIDu** y **PIDw**, las cuales toman las referencias y reasignan las variables u y w de acuerdo con la ley de control.

Finalmente se aplica las velocidades u y w asignadas por medio de la función applySpeed.

Modo 2 – Parada de emergencia

Si se indica este modo por medio del radio control, se envía la indicación a los módulos DAC de enviar un voltaje de 0.0V y se activa la salida de parada de emergencia.

3.7 Declaración de las funciones

En esta subsección se detallan las funciones implementadas en Ceres, que son usadas en el loop previamente mostrado.

3.7.1 Applycontrol

Esta función es la configurada para el timer, y será ejecutada cada ControlTime (en la sección 2 de parámetros), el cual esta configurado a 20ms actualmente y se ejecuta cuando se configura la bandera closeloop a true.

Esta función toma las referencias de acuerdo con el modo actual del vehículo (radio control si es manual o ROS si es automático), y calcula la señal de control haciendo uso de las funciones **PIDu** y **PIDw**, finalmente aplica las señales de control con la función **applySpeed.**

3.7.2 receiveRefPose

Esta función es un callback que se ejecuta cada vez que hay un mensaje en el topic /**ceres/cmd_vel**, su ejecución permite extraer la velocidad lineal y angular deseada a variables globales que serán usadas en el control.

3.7.3 receivePose

Esta función es un callback que se ejecuta cada vez que hay un mensaje en el topic **/advanced_navigation_driver/twist2D**, su ejecución permite extraer la velocidad lineal y angular actual del robot Ceres a variables globales que serán usadas en el control.

3.7.4 receiveRC

Esta función es un callback que se ejecuta cada vez que hay un mensaje en el topic **/ceres/RC** publicado por el Arduino nano que procesa información del radio control, su ejecución permite extraer los valores de cada canal del radio control a variables globales que serán usadas la lógica del programa.

Finalmente, la función asigna el modo de operación según el valor de los canales de emergencia y auxiliar del radio control.

3.7.5 PIDu

La función recibe una referencia de velocidad lineal y calcula la salida de un controlador PID usando la medición de la IMU.

Nota: Su implementación es único cambio entre los dos programas que se adjuntan.

Luego de calcular la salida del controlador se realiza **una saturación a +-0.8 m/s** y se activa la bandera de saturación si es el caso, de acuerdo con el valor de la bandera de saturación se actualizan los valores de las variables del control **lastUerr** y **IUerror.**

3.7.6 PIDw

La función recibe una referencia de velocidad angular y calcula la salida de un controlador PID usando la medición de la IMU.

Nota: Su implementación es único cambio entre los dos programas que se adjuntan.

Luego de calcular la salida del controlador se realiza **una saturación a +-0.3 rad/s** y se activa la bandera de saturación si es el caso, de acuerdo con el valor de la bandera de saturación se actualizan los valores de las variables del control **lastWerr y IWerror**.

3.7.7 applySpeed

Esta función recibe comandos de velocidad lineal y angular, con lo cual calcula los voltajes a aplicar a los drivers mediante una función de transferencia de primer orden.

Inicialmente valida el rango de los comandos recibidos de acuerdo con maxVel y maxAngVel definido en los parámetros.

En este punto se tiene una sección debidamente comentada que llena los datos del mensaje a enviar por el topic /ceres/controlSignal, con las señales de control (en velocidad lineal y angular) que se enviaran al robot.

Se realiza la conversión de las velocidades deseadas angular y lineal a las velocidades de las ruedas, teniendo en cuenta las dimensiones del robot.

A continuación, se definen y asignan las variables Ul y Ur las cuales van a ser los voltajes a aplicar a los drivers, para su conversión se aplica la siguiente formula:

Ul=((30*Wl)/3.14)/Kl+Umin;

Ur=((30*Wr)/3.14)/Kr+Umin;

Donde Kr y Kl son coeficientes medidos mediante pruebas repetitivas con los drivers y Umin es el voltaje mínimo para enviar al driver a partir del cual se empieza a generar movimiento.

Posteriormente se realiza la validación para que los voltajes calculados estén dentro del rango máximo y mínimo del driver.

Se realizan las conversiones finales de los voltajes calculados a los incrementos del modulo DAC disponible y se llama la función DAC_set para que sean generados los voltajes.

3.7.8 DAC_set

Esta función recibe el valor de incrementos y el numero del DAC a enviar ese numero de incrementos, esto último pues los DAC se encuentran por comunicación I2C lo cual permite que esta función active cualquiera de los dos DAC.

Nota: En la carpeta se deja un archivo de texto plano donde se encuentra un método para el cambio del sentido de giro de las llantas dejando dos segundos para evitar malfuncionamientos o daños en el driver. Este código brinda información de como implementar una trayectoria con cambio de sentido de giro si es de interés en el futuro.