INSTITUTO DE SISTEMAS E ROBÓTICA
UNIVERSIDADE DE COIMBRA

Relatório de projeto

ROS Node FOCBOX Unity Differential controller

Implementação de camada de controlo para o controlador de motores Focbox unity

Rafael Silva - 2020

Índice

Introdução

O ROS FOCBOX Unity Differential controller, é um controlador para o framework ROS, baseado no ros_differrential_controler, que permite o controlo de robos de tração diferencial que utilizem o controlador FOCBOX Unity.

O FOCBOX Unity é um controlador de motores bldc duplo, ie. Controla dois motores em simultânio, baseado no controlador open source VESC.

Image 1 - Controlador FOCBOX Unity

O VESC Project que é um projeto que ambiciona fornecer um ambiente de facil e rapida configuração para diversos tipos de motor, oferece metodos de controlo variados como, (Field Oriented Control).

ROS (Robot operating system) é um framework que visa a apoiar o desenvolvimento de software de controlo de robos de diversos tipos e

objetivos, disponibilizando um conjunto de bibliotecas, ferramentas e convenções.

Robo de tração diferencial é um robo cuja tração tem como base, duas rodas de lados opostos do mesmo, uma diferenca de velocidade entre estas permite ao robo mudar de direção.

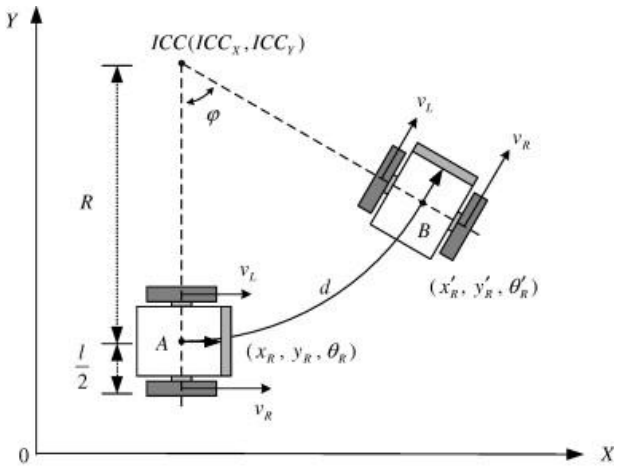

Image 2 - Robô de tração diferencial

Este documento serve apenas para uma perspetiva geral sobre o controlador, informações mais detalhadas bem como o próprio software estão disponiveis num repositório publico online.

Motivação

Surgio a necessidade de melhorar o sistema de tração do projeto INTERBOT, que utilizava motores DC numa configuração diferencial.
Motores BLDC trazem diversas vantagens em relação aos motores DC, no entanto com a utilização destes surge a necessidade de utilizar controladores mais avançados.
O FOCBOX Unity é um controlador de motores bldc duplo open-source, e bastatnte versatil, no entanto, como a sua tipica aplicação e bastante mais simples e não envolve interação com computadores, nao existe software de controlo para alem da sua ferramenta de configuração, que só está disponivel para windows, dai surge a necessidade de desenvolver este controlador.

Objetivos

O objetivo final ficou definido como implementar uma node(conceito de ROS) na framework ROS, que disponibilizasse um controlador de juntas na plataforma ROS.

- Estudar e documentar o protocolo de comunicção da VESC;
- Controlo individual de cada motor em corrente e velocidade;
- Controlo diferencial da plataforma;
- Utilização da plataforma/framework ROS;
- Captura de dados do controlador.
- Implementar node em ROS.

Controlador

O FOCBOX Unity é baseado num MCU da STMicroelectronics, o STM32F4XX, cortex-M4F.
Neste MCU corre um branch do firmware VESC, adaptado para controlar dois motores em simultâneo.
O controlador possui varias portas de comunicação, este software assume uma connecção por usb ou porta Série(UART).

Atenção, a informação seguinte é referente à versão de firmware 23.3 e pode sofrer alterações em versões seguintes.

Protocolo

O protocolo de comunicação do firmware VESC não está muito bem documentado, no entanto, o facto de ser open-source, e de existirem

ferramentas open source que utilizam este protocolo para configuração, foi possivel fazer descrição completa do protocolo.

O firmware VESC utiliza o mesmo protocolo nas suas diferentes portas de comunicação série, ie. CAN, USB e UART.

A porta CAN não foi analizada pois não era necessária para a aplicação.

A porta USB apresenta-se como uma porta série no computador(USB CDC) com o VID:PID = F055:E063, “STMicroelectronics Virtual Port”

A porta UART Utilizada um Baud Rate configuravel no firmware, com um valor predefinido de 115200.

A comunicação segue a seguinte estrutura:

1B	1:2B	1:256B	2B	1B
Start Byte	Length	Payload	CRC	Termination Byte

A comunicação é MSB first

A comunicação é sempre iniciada pelo master, ie. O controlador nunca inicia comunicação.

A comunicação é paseada em pacotes.

Um pacote é iniciado por um byte 0x02 se contiver uma <= 256 bytes ou por um byte 0x03 for > 256 bytes.

Se iniciado por 0x02 o seguinte byte é o comprimento da payload.

Se iniciado por 0x03 0s seguintes 2 bytes são o comprimento da payload.

Segue uma stream de bytes que corresponde à payload.

Seguem 2 bytes de CRC.

A comunicação termina com um byte 0x03.

Tipicamente a payload tem:

1B	1:2B
Comando	Argumentos

Alguns comandos obtêm resposta, essa respostam vem no mesmo formato que as mensagens enviadas pelo master (pacote), onde a payload contem a resposta, tipicamente:

1B	1:2B
Comando	Resposta

Feedback

O Firmware VESC disponibiliza alguns dados sobre o estado do controlador, ie. Diferencial de pulsos do motor desde que este foi inicializado, a tensão de alimentação, a corrente do motor, potencia consumida, velocidade, entre outros.

Estes dados podem ser obtidos atraves de uma transação com o controlador, enviado como payload o comando referente ao pedido de dados (pode variar com a versão de firmware), seguido de uma mask de bits que determinam os dados de resposta(não suportado por todas as versões de firmware), ou um comando que pede todos os dados disponiveis.

O controlador responde com um pacote onde a payload contem o comando, e os dados pedidos ordenadamente(ordem não muda).

Apesar de a payload ser composta por unsigned bytes, os dados vêm em varios formatos e têm de ser convertidos.

Algumas particularidades:

O controlador reporta 90 pulsos por rotação de motor (testado apenas com hall effects).

O controlador reporta velocidade em ERPM (eletrical rpm), ERPM = RPM * PP(pole pairs).

Para obter rpm dividimos erpm pelo numero de pares de polos do motor.

Para obter ω (velocidade angular) temos que $\omega = 2\pi * (rpm / 60)$.

O controlador não suporta reset dos pulsos de encoder no firmware original, no entanto é posivel implementar com uma simples modificação, isto é necessário para que o controlador seja initicializado no estado conhecido.

Exemplo de pacote

Comando para definir a corrente dos motores.

Start Byte	Length	Payload						CRC	Termination Byte
	6	Comando	Motor	Valor float 32bit					
0x02	0x06	0x07	0x01	0x00	0x00	0x00	0x00	0x00 10	0x03

ROS

ROS Controler

Para facilitar a integração com outros softwares existentes e futuros, o controlador usa como base as conveções e bibliotecas dos packages ROS Controllers, nestas vêm incluidas varias formas de implentar uma node de controlador, esta implementação tira proveito do controlador diff_drive_controller que possui ja uma implementação de alto nivel da odometria, necessita apenas de dois controladores de baixo nivel de velocidade, os quais o FOCBOX Unity possui internamente.

O diff_drive_controller é um controlador de velocidade, recebe um comando de velocidade linear e velocidade angular(rodar sobre si) e decompõe em velocidades para cada motor.

Atraves dos pulsos que recebemos do FOCBOX Unity que convertemos em posição angular, o diff_drive_controller determina a posição e orientatação do robo no espaço (odometria), para este efeito o controlador recebe diversos paremetros de entrada como a distancia entre rodas, seu diametro, numero de polos do motor, etc.

A velocidades retornadas pelo diff_drive_controller vêm em ω (velocidade angular) esta é convertida em ERPM e enviada para o controlador que faz o controlo de velocidade e corrente interno.

ROS Node

Uma node é um conceito de ros, é um processo que interage com outros processos e servicos da framework ros, para controlar um robo.
Esta node faz a ligação do ROS com o controlador FOCBOX Unity, bem como o controlo diferencial da plataforma onde este está instalado, também publica os dados referentes ao controlador em topicos para serem utilizados por outros processos.
A node está escrita em C++.

A thread principal do programa inicia as variaveis necessárias aos controladores ros, os proprios controladores e o control manager do ros.
Depois abre a porta série passada como parametro nos ficheiros de configuração, abre uma thread que monitoriza a entrada de dados na porta série, e, através de uma maquina de estados, identifica, monta e verifica os frames enviados pelo controlador, e tenta inicializar o controlador, enviado um comando de reset, seguido de um comando para identificar a versão de firmware do controlador, se tudo se comportar como esperado, o programa continua.
inicializa um timer do framework ros, com um periodo correspondente ao parametro de configuração. Na callback deste timer, o programa envia um comando a pedir dados de estado, ie. velocidade, corrente, etc.
Na thread de receção mencionada anteriormente a resposta é processada automaticamente, e os dados sao publicados no topico correspondente.
A thread principal fica a rodar a função ros.spin.
A framework ros faz o processamento autonomo dos seus controladores periodicamente, e chama uma funcção quando é necessário enviar dados para o FOCBOX Unity, as velocidades vêm em velocidade angular, são convertidas, e são montados e enviados pacotes de comando.

Limitações

A utilização dos controladores existentes no ROS facilitou bastante a implementação do software, no entanto está limitada a controlo de velocidade, e idealmente seria realizado um controlo por esforço, no futuro terá de ser implementado um controlador de raiz para o efeito, mas a camada de comunicação com o hardware está completamente funcional vai facilitar implementações futuras.

Utilização

As instruções e recomendações de utilização estão disponiveis no repositório publico onde está o software.

https://github.com/gimbas/focbox_unity_diff_driver
https://github.com/gimbas/focbox_unity_diff_driver/blob/master/API.md

Repositório

Este software foi desenvolvido como open-source segundo a licensa MIT, e está disponivel num repositório publico online.

https://github.com/gimbas/focbox_unity_diff_driver

Referências

- https://www.ros.org/about-ros/
- https://wiki.ros.org/ros_control
- https://wiki.ros.org/diff_drive_controller
- https://wiki.ros.org/Nodes
- https://vesc-project.com/
- https://www.enertionboards.com/
- https://github.com/gimbas/focbox_unity_diff_driver
- https://github.com/vedderb/bldc
- https://github.com/gimbas/focbox_unity_diff_driver/blob/master/API.md