

Relatório de projeto

ROS FOCBOX Unity Differential controller node

Implementação de camada de controlo para o controlador de motores FOCBOX Unity

Índice

Controlador.....	6
Exemplo de pacote.....	7
Feedback.....	7
Protocolo.....	6
Feedback.....	7
Introdução.....	3
Limitações.....	9
Motivação.....	4
Objetivos.....	4
Protocolo.....	6
Referências.....	10
Repositório.....	9
ROS.....	8
ROS Controler.....	8
ROS Node.....	8
Utilização.....	9
Índice.....	2

Introdução

O ROS FOCBOX Unity Differential controller node, é uma *node*/controlador para o *framework* ROS, que permite o controlo de plataformas de tração diferencial que utilizem o controlador FOCBOX Unity.

O FOCBOX Unity é um controlador de motores BLDC duplo, ie. Controla dois motores em simultâneo, baseado no controlador *open-source* VESC.



Image 1 - Controlador FOCBOX Unity

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

ROS (Robot operating system) é um *framework* que visa a apoiar o desenvolvimento de software de controlo de robôs de diversos tipos e objetivos, disponibilizando um conjunto de bibliotecas, ferramentas e convenções.

Robô de tração diferencial é um robô cuja tração tem como base, duas rodas de lados opostos do mesmo, uma diferença de velocidade entre estas permite a mudança de direção.

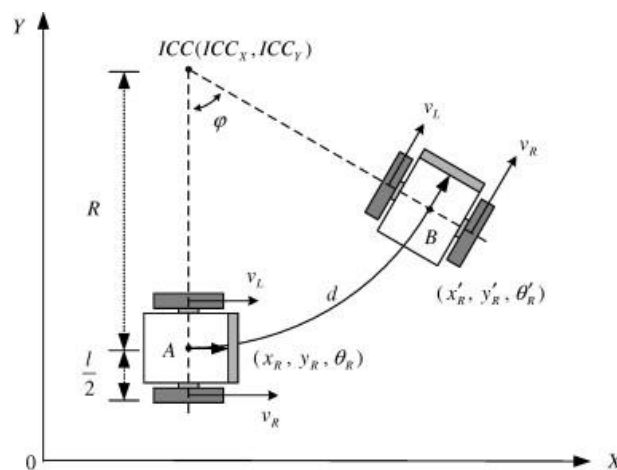


Image 2 - Robô de tração diferencial

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

Motivação

Surgiu 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 bastante versátil, no entanto, como a sua típica aplicação é bastante mais simples e não envolve interação com computadores, não existe software de controlo para além da sua ferramenta de configuração, que só está disponível para Windows, daí 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 e ou permitisse o controlo diferencial.

- Estudar e documentar o protocolo de comunicação 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 conexão por USB ou porta Série(UART).

Atenção, a informação seguinte é referente à versão de *firmware* 23.30 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 também *open-source* que utilizam o protocolo para configuração, foi possível 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 analisada 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* configurável 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 baseada 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 Os 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, que vêm 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 através de uma transação com o controlador, enviando 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 disponíveis.

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 vários 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 (electrical 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 possível implementar com uma simples modificação, isto é necessário para que o controlador seja inicializado no estado conhecido. (uma versão com esta modificação está disponível).

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 Controller

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

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

Através dos pulsos que recebemos do FOCBOX Unity que convertemos em posição angular, o *ros_diff_driver* determina a posição e orientação do robô no espaço (odometria), para este efeito o controlador recebe diversos parâmetros de entrada como a distancia entre rodas, seu diâmetro, numero de polos do motor, etc.

A velocidades retornadas pelo *ros_diff_driver* vêm em velocidade angular(ω) 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 serviços da *framework* ROS, para controlar um robô.

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 tópicos para serem utilizados por outros processos.

A *node* está escrita em C++.

A *thread* principal do programa inicia as variáveis necessárias aos controladores ROS, os próprios controladores e o *control manager* do ROS.

Depois abre a porta série passada como parâmetro nos ficheiros de configuração, abre uma *thread* que monitoriza a entrada de dados na porta, e, através de uma máquina de estados, identifica, monta e verifica os *frames* recebidas do controlador. 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 período correspondente ao parâmetro 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 são publicados no tópico correspondente.

A *thread* principal fica a rodar a função `ros.spin`, a *framework* ROS faz o processamento autónomo dos seus controladores periodicamente, e chama uma função 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 disponíveis 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 licença MIT, e está disponível 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