Drive Car

1

Generated by Doxygen 1.8.13

Contents

Chapter 1

Drive Car

1.1 Introduction

This project aims to drive a car from a Linux pc using a game controller.
For communication with the car interfaces, a comma.ai Panda is used.
The comma.ai Panda is talked to via USB and the libusb.

1.2 Build

To build this project, you can just run
`make`

To clean all the build files and the compiled software, run
`make clean`

Chapter 2

Drive-Car

This is some code to control a Toyota Rav4 Hybrid using a Linux PC.

To control the car a Panda is used. This panda is being communicated with using the libusb library.

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Data Structure Documentation

5.1 axisState Struct Reference

Data Fields

- int16_t **x**
- int16_t **y**

The documentation for this struct was generated from the following file:

- joystick.h

5.2 CANFrame Struct Reference

Defines a standard CAN frame.

```
#include <panda.h>
```

Data Fields

- uint16_t ID

  The CAN frame ID.
- uint8_t data [8]

  The Data sent with the frame, max. 8 Bytes.
- uint8_t bus

  Which bus to send the data on. For using multiple CAN busses.
- uint8_t length

  The number of bytes te be sent.
- uint8_t freq

  How frequent to send the frame.

5.2.1 Detailed Description

Defines a standard CAN frame.

This struct defines a standard CAN frame, so that the software can be used with different CAN devices with different drivers.

The documentation for this struct was generated from the following file:

- panda.h

5.3 CARState Struct Reference

Data Fields

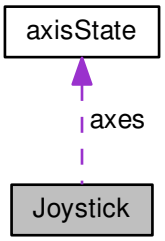
- float **speed**
- float **angle**

The documentation for this struct was generated from the following file:

- main.c

5.4 Joystick Struct Reference

Collaboration diagram for Joystick:



Data Fields

- const char * **name**
- int **fd**
- axisState **axes** [3]
- uint8_t **buttons** [12]
- uint8_t **numberOfAxes**
- uint8_t **numberOfButtons**

The documentation for this struct was generated from the following file:

- joystick.h

5.5 Panda Struct Reference

Defines the interface for a specific connected Panda.

`#include <panda.h>`

Data Fields

- libusb_device_handle * handle
  The LibUSB handle.
- struct libusb_device_descriptor desc
  The LibUSB file descriptor.

5.5.1 Detailed Description

Defines the interface for a specific connected Panda.

This struct contains the USB handle and file descriptor, so it can be passed to all functions.

The documentation for this struct was generated from the following file:

- panda.h

5.6 Params Struct Reference

Data Fields

- char * **js**
- uint8_t **enableDsu**
- uint8_t **enableCam**

The documentation for this struct was generated from the following file:

- main.c

Chapter 6

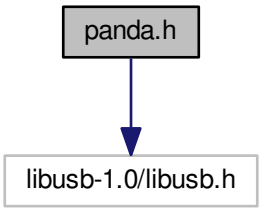
File Documentation

6.1 panda.h File Reference

File containing all panda specific function declarations.

```
#include <libusb-1.0/libusb.h>
```

Include dependency graph for panda.h:



Data Structures

- struct Panda

  Defines the interface for a specific connected Panda.
- struct CANFrame

  Defines a standard CAN frame.

Macros

- #define REQUEST_IN (LIBUSB_ENDPOINT_IN | LIBUSB_REQUEST_TYPE_VENDOR | LIBUSB_RECI↩ PIENT_DEVICE)

  Constant to define what type of request you want to make.
- #define REQUEST_OUT (LIBUSB_ENDPOINT_OUT | LIBUSB_REQUEST_TYPE_VENDOR | LIBUSB_↩ RECIPIENT_DEVICE)

  Constant to define what type of request you want to make.

Functions

- int panda_setup (Panda *p)

  Setup and connect to the Panda.
- int panda_connect (Panda *p)

  Connect to the Panda (Called from setup)
- int panda_close (Panda *p)

  Close the USB handle of the Panda.
- int panda_get_version (Panda *p)

  Retrieve and print the current version of the Panda firmware.
- int panda_set_safety_mode (Panda *p, uint16_t mode)

  Set the safety mode of the Panda, to allow sending on the CAN busses.
- int panda_set_can_speed (Panda *p, int bus, int speed)

  Set the speed of a specific CAN bus of the Panda.
- int panda_can_send_many (Panda *p, CANFrame frames[ ], int length)

  Send many CAN frames to the Panda.
- int panda_can_send (Panda *p, CANFrame frame)

  Send one CAN frame to the Panda.
- int panda_can_recv (Panda *p, unsigned char *data, int length)

  Request received CAN frames from the Panda.
- int panda_can_clear (Panda *p, int bus)

  Clear an internal buffer of the Panda.
- void print_many (CANFrame frames[ ], int length)

  Debug the frames that would be sent.

6.1.1 Detailed Description

File containing all panda specific function declarations.

Author

Laurens Wuyts

Date

10 May 2018 This file contains all the function declarations for using the panda, as well as the definition of the Panda struct.

6.1.2 Macro Definition Documentation

6.1.2.1 REQUEST_IN

```
#define REQUEST_IN (LIBUSB_ENDPOINT_IN | LIBUSB_REQUEST_TYPE_VENDOR | LIBUSB_RECIPIENT_DEVICE)
```

Constant to define what type of request you want to make.

These defines are constants to define what type of USB request is made.

6.1.2.2 REQUEST_OUT

```
#define REQUEST_OUT (LIBUSB_ENDPOINT_OUT | LIBUSB_REQUEST_TYPE_VENDOR | LIBUSB_RECIPIENT_DEVI↩
CE)
```

Constant to define what type of request you want to make.

These defines are constants to define what type of USB request is made.

6.1.3 Function Documentation

6.1.3.1 panda_can_clear()

```
int panda_can_clear (
            Panda * p,
            int bus )
```

Clear an internal buffer of the Panda.

Parameters

p	Pointer to Panda struct.
bus	The bus to clear the buffer of.

Returns

0 Success
< 0 Fail

6.1.3.2 panda_can_recv()

```
int panda_can_recv (
            Panda * p,
            unsigned char * data,
            int length )
```

Request received CAN frames from the Panda.

Parameters

p	Pointer to Panda struct.
data	The received data from the Panda.
length	The maximum quantity of data to request.

Returns

0 Success
< 0 Fail

6.1.3.3 panda_can_send()

```
int panda_can_send (
            Panda * p,
            CANFrame frame )
```

Send one CAN frame to the Panda.

Parameters

p	Pointer to Panda struct.
frame	The frame to send.

Returns

0 Success
< 0 Fail

6.1.3.4 panda_can_send_many()

```
int panda_can_send_many (
            Panda * p,
            CANFrame frames[],
            int length )
```

Send many CAN frames to the Panda.

Parameters

p	Pointer to Panda struct.
frames	The CAN frames to send to the Panda.
length	The number of CAN frames to send.

Returns

0 Success
< 0 Fail

6.1.3.5 panda_close()

```
int panda_close (
            Panda * p )
```

Close the USB handle of the Panda.

Parameters

p	Pointer to Panda struct.

Returns

0 Success
< 0 Fail

6.1.3.6 panda_connect()

```
int panda_connect (
            Panda * p )
```

Connect to the Panda (Called from setup)

Parameters

p	Pointer to Panda struct.

Returns

0 Success
< 0 Fail

6.1.3.7 panda_get_version()

```
int panda_get_version (
            Panda * p )
```

Retrieve and print the current version of the Panda firmware.

Parameters

p	Pointer to Panda struct.

Returns

0 Success
< 0 Fail

6.1.3.8 panda_set_can_speed()

```
int panda_set_can_speed (
            Panda * p,
            int bus,
            int speed )
```

Set the speed of a specific CAN bus of the Panda.

Parameters

p	Pointer to Panda struct.
bus	Which bus to change
speed	The speed to set in kbps

Returns

0 Success
< 0 Fail

6.1.3.9 panda_set_safety_mode()

```
int panda_set_safety_mode (
            Panda * p,
            uint16_t mode )
```

Set the safety mode of the Panda, to allow sending on the CAN busses.

Parameters

p	Pointer to Panda struct.
mode	Mode to set the Panda to. (0 = listen only, 0x1337 = Write all)

Returns

0 Success
< 0 Fail

6.1.3.10 panda_setup()

```
int panda_setup (
            Panda * p )
```

Setup and connect to the Panda.

Parameters

p	Pointer to Panda struct.

Returns

0 Success
< 0 Fail

6.1.3.11 print_many()

```
void print_many (
            CANFrame frames[],
            int length )
```

Debug the frames that would be sent.

Parameters

frames	The frames to print.
length	The number of frames to print.

Returns

0 Success
< 0 Problem

Index