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 Axis Struct Reference

Contains the X and Y value of an axis.

```
#include <joystick.h>
```

Data Fields

- int16_t x

  The X value of a joystick axis.
- int16_t y

  The Y value of a joystick axis.

5.1.1 Detailed Description

Contains the X and Y value of an axis.

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 Health Struct Reference

Contains a few health parameters of the car and the Panda.

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

Data Fields

- uint32_t voltage

  The car power voltage.
- uint32_t current

  The current drawn by the Panda.
- uint8_t started

  Is the car started?
- uint8_t controls_allowed

  Is it allowed to control the car?
- uint8_t **gas_interceptor_detected**
- uint8_t started_signal_detected

  (Deprecated) Not used anymore
- uint8_t started_alt

  (Deprecated) Not used anymore

5.3.1 Detailed Description

Contains a few health parameters of the car and the Panda.

This struct contains a few health parameters of the car and the Panda.

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

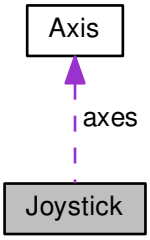
- panda.h

5.4 Joystick Struct Reference

Defines the interface for a connected joystick/gamepad.

`#include <joystick.h>`

Collaboration diagram for Joystick:



Data Fields

- const char * name

  The Name of the interface.
- int fd

  The file desctriptor to read from and write to.
- Axis axes [3]

  The values of all axes of the gamepad.
- uint8_t buttons [12]

  The state of all buttons on the joystick/gamepad.
- uint8_t numberOfAxes

  The number of axes that the specific joystick has.
- uint8_t numberOfButtons

  The number of buttons that a specific joystick has.

5.4.1 Detailed Description

Defines the interface for a connected joystick/gamepad.

This struct contains the definition of a joystick or gamepad so it can be passed between functions.

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 joystick.h File Reference

File containing the library to use a gamepad in Linux.

Data Structures

- struct Axis

  Contains the X and Y value of an axis.
- struct Joystick

  Defines the interface for a connected joystick/gamepad.

Functions

- int setupJoystick (Joystick *js, char *name)

  Setup and connect to a joystick.
- int readJoystick (Joystick *js)

  Reads te current status of the joystick and puts it in the struct.
- void printState (Joystick *js, int enableAxes, int enableButtons)

  Debugs the status of the joystick. Can be configured to show only the axes, the buttons or both.

6.1.1 Detailed Description

File containing the library to use a gamepad in Linux.

Author

Laurens Wuyts

Date

28 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 Function Documentation

6.1.2.1 printState()

```
void printState (
            Joystick * js,
            int enableAxes,
            int enableButtons )
```

Debugs the status of the joystick. Can be configured to show only the axes, the buttons or both.

Parameters

js	Pointer to Joystick struct.
enableAxes	Prints the values of the axes.
enableButtons	Prints the values of all the buttons.

6.1.2.2 readJoystick()

```
int readJoystick (
            Joystick * js )
```

Reads te current status of the joystick and puts it in the struct.

Parameters

js	Pointer to Joystick struct.

Returns

0: Success
<0: Fail

6.1.2.3 setupJoystick()

```
int setupJoystick (
            Joystick * js,
            char * name )
```

Setup and connect to a joystick.

Parameters

js	Pointer to Joystick struct.
name	The name of the joystick to connect to.

Returns

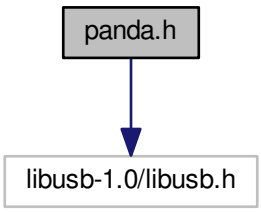
0: Success
<0: Fail

6.2 panda.h File Reference

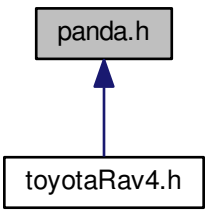
File containing all panda specific function declarations.

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

Include dependency graph for panda.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct Panda

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

  Defines a standard CAN frame.
- struct Health

  Contains a few health parameters of the car and the Panda.

Macros

- #define REQUEST_IN (LIBUSB_ENDPOINT_IN | LIBUSB_REQUEST_TYPE_VENDOR | LIBUSB_RECIPIENT_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, int mode)
- 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_get_health** (Panda *p, Health *h)
- 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.

6.2.1 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.

6.2.2 Macro Definition Documentation

6.2.2.1 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.

6.2.2.2 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.

6.2.3 Function Documentation

6.2.3.1 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

6.2.3.2 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

6.2.3.3 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

6.2.3.4 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

6.2.3.5 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

6.2.3.6 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

6.2.3.7 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

6.2.3.8 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

6.2.3.9 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

6.2.3.10 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

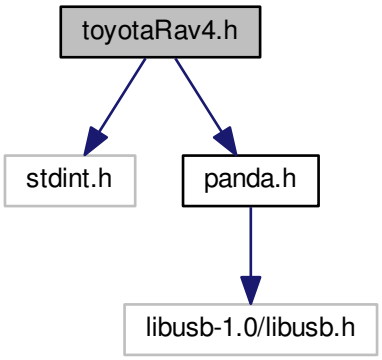
0: Success
<0: Problem

6.3 toyotaRav4.h File Reference

File containing all Toyota Rav4 specific function declarations.

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

Include dependency graph for toyotaRav4.h:



Macros

- #define **ARRAY_LENGTH**(arr) (sizeof(arr) / sizeof((arr)[0]))

Functions

- uint16_t create_checksum (CANFrame *frame)
  Calculate the checksum of the CAN frame.
- int sendStaticVideo (CANFrame frames[ ], uint16_t count)
  Send the static messages to replace the video from the camera.
- int sendStaticCam (CANFrame frames[ ], uint16_t count)
  Send the static messages to replace the camera.
- int sendStaticDsu (CANFrame frames[ ], uint16_t count)
  Send the static messages to replace the DSU.
- int sendSteerCommand (CANFrame frames[ ], uint16_t count, uint16_t torque)
  Send the message to control the steering wheel.
- int sendAccelCommand (CANFrame frames[ ], uint16_t count, uint16_t acceleration, uint8_t cancel)
  Send the message to control the acceleration and braking of the car.
- int sendUiCommand (CANFrame frames[ ], uint16_t count, uint8_t status)
  Send the messages to control the heads up display.
- int sendFcwCommand (CANFrame frames[ ], uint16_t count, uint8_t fcw)
  Send the message to enable or disable Forward Collision Warning.

6.3.1 Detailed Description

File containing all Toyota Rav4 specific function declarations.

Author

Laurens Wuyts

Date

28 May 2018 This file contains all the function declarations for send all the right CAN bus messages for controlling a Toyota Rav4 Hybrid.

6.3.2 Function Documentation

6.3.2.1 create_checksum()

```
uint16_t create_checksum (
            CANFrame * frame )
```

Calculate the checksum of the CAN frame.

Parameters

frame	The frame to calculate the checksum for.

Returns

The calculated checksum.

6.3.2.2 sendAccelCommand()

```
int sendAccelCommand (
            CANFrame frames[],
            uint16_t count,
            uint16_t acceleration,
            uint8_t cancel )
```

Send the message to control the acceleration and braking of the car.

Parameters

frames	The array to add the messages to.
count	The 100Hz counter of the program.
acceleration	The force to accelerate or decelerate with. (Negative is decelerate)
cancel	Bit to cancel the controls and turn of cruise control.

Returns

Number of messages added.

6.3.2.3 sendFcwCommand()

```
int sendFcwCommand (
            CANFrame frames[],
```

```
            uint16_t count,
            uint8_t fcw )
```

Send the message to enable or disable Forward Collision Warning.

Parameters

frames	The array to add the messages to.
count	The 100Hz counter of the program.
fcw	Enable/Disable the Forward Collision Warning.

Returns

Number of messages added.

6.3.2.4 sendStaticCam()

```
int sendStaticCam (
            CANFrame frames[],
            uint16_t count )
```

Send the static messages to replace the camera.

Parameters

frames	The array to add the messages to.
count	The 100Hz counter of the program.

Returns

Number of messages added.

6.3.2.5 sendStaticDsu()

```
int sendStaticDsu (
            CANFrame frames[],
            uint16_t count )
```

Send the static messages to replace the DSU.

Parameters

frames	The array to add the messages to.
count	The 100Hz counter of the program.

Returns

Number of messages added.

6.3.2.6 sendStaticVideo()

```
int sendStaticVideo (
            CANFrame frames[],
            uint16_t count )
```

Send the static messages to replace the video from the camera.

Parameters

frames	The array to add the messages to.
count	The 100Hz counter of the program.

Returns

Number of messages added.

6.3.2.7 sendSteerCommand()

```
int sendSteerCommand (
            CANFrame frames[],
            uint16_t count,
            uint16_t torque )
```

Send the message to control the steering wheel.

Parameters

frames	The array to add the messages to.
count	The 100Hz counter of the program.
torque	The amount of torque to add to the steering wheel.

Returns

Number of messages added.

Hud: ∗ 0x00 - Regular ∗ 0x40 - Actively Steering (beep) ∗ 0x80 - Actively Steering (no beep) ∗

6.3.2.8 sendUiCommand()

```
int sendUiCommand (
            CANFrame frames[],
```

```
            uint16_t count,
            uint8_t status )
```

Send the messages to control the heads up display.

Parameters

frames	The array to add the messages to.
count	The 100Hz counter of the program.
status	The status of the heads up display.

Returns

Number of messages added.

Index