My Project

Generated by Doxygen 1.8.11

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

Class Documentation

3.1 control_loop Class Reference

`#include <control_loop.h>`

3.1.1 Detailed Description

Created by nate furbeyre on 11/18/18. The control loop task runs the control loops for both the steering servo and the motor driver This class has not been fully implemented due to the time constraints of the class.

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

- control_loop.h

3.2 fifth_wheel Class Reference

`#include <fifth_wheel.h>`

Inheritance diagram for fifth_wheel:

3.3 gear_shifter Class Reference

`#include <gear_shifter.h>`

Inheritance diagram for gear_shifter:

Collaboration diagram for gear_shifter:

Public Member Functions

- gear_shifter (const char *a_name, unsigned char a_priority=0, size_t a_stack_size=configMINIMAL_STA↩ CK_SIZE, emstream *p_ser_dev=NULL, semi_truck_data_t *semi_data_in=NULL)

  The constructor for a gear_shifter that handles gear shifting of the semi-truck.
- void run ()

  Runs the task code for the steering servo. This method simulates finite state machine with 2 different states: on (1) and off (2). In its off state, the servo will not try to actuate to any specific steering angle, it will simply remain where it is when it was turned off. In its on state, the servo will actuate (based on PWM level) to hit the setpoint steering angle from the Raspberry Pi's control loop task.
- void **shift_to_first** ()
- void **shift_to_second** ()
- void **shift_to_third** ()
- uint8_t get_actual_level ()

  gets the level of the gear shifter to potentially let other tasks know what gear the truck is in.
- void set_desired_level (uint8_t in_level)

  Sets the desired level of the gear shifter based on input from the RC controller.

3.3.1 Detailed Description

Created by nate furbeyre on 11/18/18. The gear shifter is responsible for shifting gears for the semi-truck. It is implemented as a child class of both a Servo, and as a TaskBase, allowing it to inherit properties and methods from each parent class.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 gear_shifter::gear_shifter (const char * *a_name*, unsigned char *a_priority* = `0`, size_t *a_stack_size* = `configMINIMAL_STACK_SIZE`, emstream * *p_ser_dev* = `NULL`, semi_truck_data_t * *semi_data_in* = `NULL`)

The constructor for a gear_shifter that handles gear shifting of the semi-truck.

Parameters

a_name	the name of the task
a_priority	The priority given to this task
a_stack_size	The amount of bytes given to the task
p_ser_dev	A serial device that this tasks output is sent to
semi_data↩ _in	A pointer to the semi truck system data communicated between tasks

3.3.3 Member Function Documentation

3.3.3.1 uint8_t gear_shifter::get_actual_level ( )

gets the level of the gear shifter to potentially let other tasks know what gear the truck is in.

Returns

the gear level that the truck is in (either 1, 2 or 3)

3.3.3.2 void gear_shifter::set_desired_level (uint8_t *in_level*)

Sets the desired level of the gear shifter based on input from the RC controller.

Parameters

in_level	The gear level that the servo needs to be actuated to

The documentation for this class was generated from the following files:

- gear_shifter.h
- gear_shifter.cpp

3.4 imu_task Class Reference

Inheritance diagram for imu_task:

Collaboration diagram for imu_task:

Public Member Functions

- imu_task (const char *a_name, unsigned char a_priority=0, size_t a_stack_size=configMINIMAL_STACK_SIZE, emstream *p_ser_dev=NULL, uint32_t sensorID=-1, uint8_t address=BNO055_ADDRESS_A, semi_truck_data_t *semi_data_in=NULL)

  The constructor for a mega_comm_task object communicates with the Rasp-Pi.
- void run ()

3.4.1 Constructor & Destructor Documentation

3.4.1.1 imu_task::imu_task (const char * *a_name*, unsigned char *a_priority* = `0`, size_t *a_stack_size* = `configMINIMAL_STACK_SIZE`, emstream * *p_ser_dev* = `NULL`, uint32_t *sensorID* = `-1`, uint8_t *address* = `BNO055_ADDRESS_A`, semi_truck_data_t * *semi_data_in* = `NULL`)

The constructor for a mega_comm_task object communicates with the Rasp-Pi.

Parameters

a_name	the name of the task
a_priority	The priority given to this task
a_stack_size	The amount of bytes given to the task
p_ser_dev	A serial device that this tasks output is sent to
sensorID	The ID number for the IMU sensor
address	The actual address for the BNO055 device itself
semi_data_in	A pointer to the semi truck system data that is communicated between tasks

3.4.2 Member Function Documentation

3.4.2.1 void imu_task::run ( )

Runs the infinite loop task code for reading IMU data. This task has only one effective state; reading angle data of the semi truck to be input into the control loop for a steering servo output.

The documentation for this class was generated from the following files:

- imu_task.h
- imu_task.cpp

3.5 mega_comm_task Class Reference

`#include <mega_comm_task.h>`

Inheritance diagram for mega_comm_task:

Collaboration diagram for mega_comm_task:

Public Member Functions

- mega_comm_task (const char *a_name, unsigned portBASE_TYPE a_priority=0, size_t a_stack_↩ size=configMINIMAL_STACK_SIZE, emstream *p_ser_dev=NULL, uint16_t baud=9600, uint8_t port=0, communication_data *comm_data_in=NULL)

  The constructor for a mega_comm_task object communicates with the Rasp-Pi.
- void run ()

  Runs the code for the ATMega64 to transmit and receive data from the Raspberry Pi This method runs on an infinite loop where data is read via an RS232 port from the raspberry pi. The data, which contains information regarding different desired values from the user and control loop, is then relayed to each of the different tasks within the A↩ TMega, so that each task can act accordingly. In addition, this method sends information back to the Raspberry Pi about the actual state of each of the tasks in the ATMega.
- void read_from_pi ()

  Reads data from the raspberry pi through one of the USART ports of the ATMega. Since this class descents the rs232int class, it is able to use its communication based methods for talking with the Raspberry Pi.
- void write_to_pi ()

  Writes data to the raspberry pi through one of the USART ports of the ATMega. Since this class descents the rs232int class, it is able to use its communication based methods for talking with the Raspberry Pi.
- void write_16bit_val (int16_t write_val)

  writes a 16 bit value to the rs232 port. This function is implemented using calls to the rs232 class getchar, along with some bitshifting to get a final value
- int16_t read_16bit_val ()

  reads a 16 bit balue from the rs232 port. This function is implemented using calls to the rs232 class getchar, along with some bitshifting to get a final value

3.5.1 Detailed Description

Created by nate furbeyre on 11/18/18. The mega_comm_task task is responsible for receiving information from the raspberry pi and relaying the necessary information to each of the tasks controlled by the ATMega64. In addition, it sends information back to the Raspberry Pi about different states of tasks on the ATMega. It uses the rs232 communication protocol and the UART/ USART ports on the ATMega.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 mega_comm_task::mega_comm_task (const char * *a_name*, unsigned portBASE_TYPE *a_priority* = 0, size_t *a_stack_size* = configMINIMAL_STACK_SIZE, emstream * *p_ser_dev* = NULL, uint16_t *baud* = 9600, uint8_t *port* = 0, communication_data * *comm_data_in* = NULL)

The constructor for a mega_comm_task object communicates with the Rasp-Pi.

Parameters

a_name	the name of the task
a_priority	The priority given to this task
a_stack_size	The amount of bytes given to the task
p_ser_dev	A serial device that this tasks output is sent to
baud	The baud rate for the UART port that the mega communicates with
port	The UART port number
semi_data_in	A pointer to the semi truck system data communicated between tasks

3.5.3 Member Function Documentation

3.5.3.1 void mega_comm_task::read_from_pi ( )

Reads data from the raspberry pi through one of the USART ports of the ATMega. Since this class descents the rs232int class, it is able to use its communication based methods for talking with the Raspberry Pi.

if the other device writes all data atomically, then we can read all data (also atomically)

3.5.3.2 void mega_comm_task::run ( )

Runs the code for the ATMega64 to transmit and receive data from the Raspberry Pi This method runs on an infinite loop where data is read via an RS232 port from the raspberry pi. The data, which contains information regarding different desired values from the user and control loop, is then relayed to each of the different tasks within the ATMega, so that each task can act accordingly. In addition, this method sends information back to the Raspberry Pi about the actual state of each of the tasks in the ATMega.

receive data from pi and relay to tasks

send data about tasks to the pi

3.5.3.3 void mega_comm_task::write_to_pi ( )

Writes data to the raspberry pi through one of the USART ports of the ATMega. Since this class descents the rs232int class, it is able to use its communication based methods for talking with the Raspberry Pi.

an ugly way to do this is through bitshifting...

The documentation for this class was generated from the following files:

- mega_comm_task.h
- mega_comm_task.cpp

3.6 motor_driver Class Reference

`#include <motor_driver.h>`

Inheritance diagram for motor_driver:

Collaboration diagram for motor_driver:

Public Member Functions

- motor_driver (const char *a_name, unsigned char a_priority=0, size_t a_stack_size=configMINIMAL_STA↩ CK_SIZE, emstream *p_ser_dev=NULL, semi_truck_data_t *semi_data_in=NULL)

  The constructor for the motor driver to supply power to the motor.
- void run ()

  Runs the task code for the motor driver. This task has a two different states: (1) reading output level that comes from the control loop running on the Raspberry pi, and actuating the motor to this level, and (2) an off state where the motor will not output any torque.

3.6.1 Detailed Description

Created by nate furbeyre on 11/19/18. The motor_driver task holds the code that runs the motor for the semi-truck. The motor involved in this project is the Tekin RX8 ESC + 1550kv combo. Output levels to the motor come from the control loop running on the Raspberry Pi that is based on the data from the RC controller and the wheel speed sensors.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 motor_driver::motor_driver (const char * *a_name*, unsigned char *a_priority* = `0`, size_t *a_stack_size* = `configMINIMAL_STACK_SIZE`, emstream * *p_ser_dev* = `NULL`, semi_truck_data_t * *semi_data_in* = `NULL`)

The constructor for the motor driver to supply power to the motor.

Parameters

a_name	the name of the task
a_priority	The priority given to this task
a_stack_size	The amount of bytes given to the task
p_ser_dev	A serial device that this tasks output is sent to
semi_data↩ _in	A pointer to the semi truck system data that is communicated between tasks

The documentation for this class was generated from the following files:

- motor_driver.h
- motor_driver.cpp

3.7 pi_comm_task Class Reference

`#include <pi_comm_task.h>`

Inheritance diagram for pi_comm_task:

Collaboration diagram for pi_comm_task:

3.7.1 Detailed Description

Created by nate furbeyre on 11/19/18. The pi_comm_task task is responsible for receiving information from the ATMega and relaying the necessary information to each of the tasks controlled by the Rasp-Pi. In addition, it sends information back to the ATMega with output values from the control loop

The documentation for this class was generated from the following files:

- pi_comm_task.h
- pi_comm_task.cpp

3.8 RC_comm Class Reference

`#include <RC_comm.h>`

3.8.1 Detailed Description

Created by nate furbeyre on 11/18/18 The RC comm task handles the reception of data from the remote device that the user can control the semi truck with. It is able to change the desired output for the gear shifter task and the fifth wheel task. In addition takes in data that changes the setpoints for the motor and steering servo that are relayed to the control loop also running on the raspberry pi. This is another class that was not implemented due to the time constraints of the class.

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

- RC_comm.h

3.9 semi_truck_data_t Struct Reference

Public Attributes

- int16_t **motor_output**
- int16_t **speed_setpoint**
- int16_t **steer_output**
- int16_t **wheel_speed**
- uint16_t **imu_angle**
- int8_t **desired_gear**
- int8_t **actual_gear**
- bool **desired_5th**
- bool **actual_5th**

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

- semi_truck_data_t.h

3.10 sensor Class Reference

`#include <LiDAR_sensor.h>`

3.10.1 Detailed Description

This class governs the control over the lidar device that is used for object detection around the semi-truck. The planned model was the Hokuyo UBG-04LX-F01 Lidar (https://www.hokuyo-aut.jp/search/single.php?serial=1 but due to the time constraints of this project, none of the actual implementation was completed for this class.

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

- LiDAR_sensor.h

3.11 Servo Class Reference

Inheritance diagram for Servo:

Public Member Functions

- uint8_t **attach** (int pin, int min, int max)
- void **write** (int value)

The documentation for this class was generated from the following files:

- Servo.h
- Servo.cpp

3.12 steer_servo Class Reference

`#include <steer_servo.h>`

Inheritance diagram for steer_servo:

Collaboration diagram for steer_servo:

Public Member Functions

- steer_servo (const char *a_name, unsigned char a_priority=0, size_t a_stack_size=configMINIMAL_STA↩ CK_SIZE, emstream *p_ser_dev=NULL, semi_truck_data_t *semi_data_in=NULL)

  The constructor for a steer_servo that handles steering of the semi-truck.
- void run ()

  Runs the task code for the steering servo. This method simulates finite state machine with a single state: on. In this state, the servo will actuate (based on PWM level) to hit the setpoint steering angle from the Raspberry Pi's control loop task.

3.12.1 Detailed Description

Created by nate furbeyre on 11/18/18. The steering servo is responsible for steering the semi-truck. It is implemented as a child class of both a Servo, and as a TaskBase, allowing it to inherit properties and methods from each parent class.

3.12.2 Constructor & Destructor Documentation

3.12.2.1 steer_servo::steer_servo (const char * *a_name*, unsigned char *a_priority* = 0, size_t *a_stack_size* = configMINIMAL_STACK_SIZE, emstream * *p_ser_dev* = NULL, semi_truck_data_t * *semi_data_in* = NULL)

The constructor for a steer_servo that handles steering of the semi-truck.

Parameters

a_name	the name of the task
a_priority	The priority given to this task
a_stack_size	The amount of bytes given to the task
p_ser_dev	A serial device that this tasks output is sent to
semi_data_in	A pointer to the semi truck system data that is communicated between tasks

The documentation for this class was generated from the following files:

- steer_servo.h
- steer_servo.cpp

3.13 wheel_speed Class Reference

`#include <wheel_speed.h>`

Inheritance diagram for wheel_speed:

Collaboration diagram for wheel_speed:

Public Member Functions

- wheel_speed (const char *a_name, unsigned char a_priority=0, size_t a_stack_size=configMINIMAL_STACK_SIZE, emstream *p_ser_dev=NULL, semi_truck_data_t *semi_data_in=NULL)

  The constructor for the wheel speed sensor to track speed of the semi-truck.
- void run ()

  Runs the task code for the wheel speed sensor. This task has a single state: reading the wheel speed sensors to determine how fast the truck is going.

3.13.1 Detailed Description

Created by nate furbeyre on 11/19/18. The wheel_speed class holds the software behind the QRE1113 Analog IR Sensor that is used to track wheel speed of the semi-truck. On the inner rim of each of the front wheels, there is a strip of alternating black and white tape. The sensor can read which color tape is measured based on the analog signal, and the data can be processed into a speed of the truck.

Some code of this class is based off of http://bildr.org/2011/06/qre1113-arduino/

3.13.2 Constructor & Destructor Documentation

3.13.2.1 wheel_speed::wheel_speed (const char * *a_name*, unsigned char *a_priority* = 0, size_t *a_stack_size* = configMINIMAL_STACK_SIZE, emstream * *p_ser_dev* = NULL, semi_truck_data_t * *semi_data_in* = NULL)

The constructor for the wheel speed sensor to track speed of the semi-truck.

Parameters

a_name	the name of the task
a_priority	The priority given to this task
a_stack_size	The amount of bytes given to the task
p_ser_dev	A serial device that this tasks output is sent to
semi_data_in	A pointer to the semi truck system data that is communicated between tasks

The documentation for this class was generated from the following files:

- wheel_speed.h
- wheel_speed.cpp