Unresolved directive in Arduino.adoc - include::../Header.adoc[] === Arduino

Version history

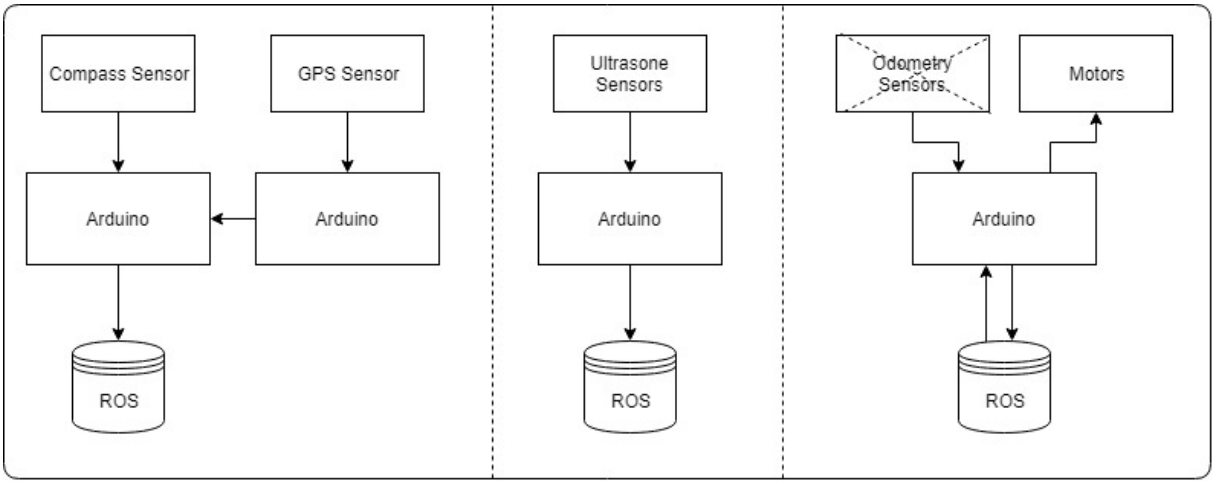
Version	Date	Person	Note
V0.1	08-03-18	Gerard Zeeman	Created Document
V1.0	09-03-18	Gerard Zeeman	Finished SAD

Preface

The following definition list is to clear up confusion about certain terms used in this document.

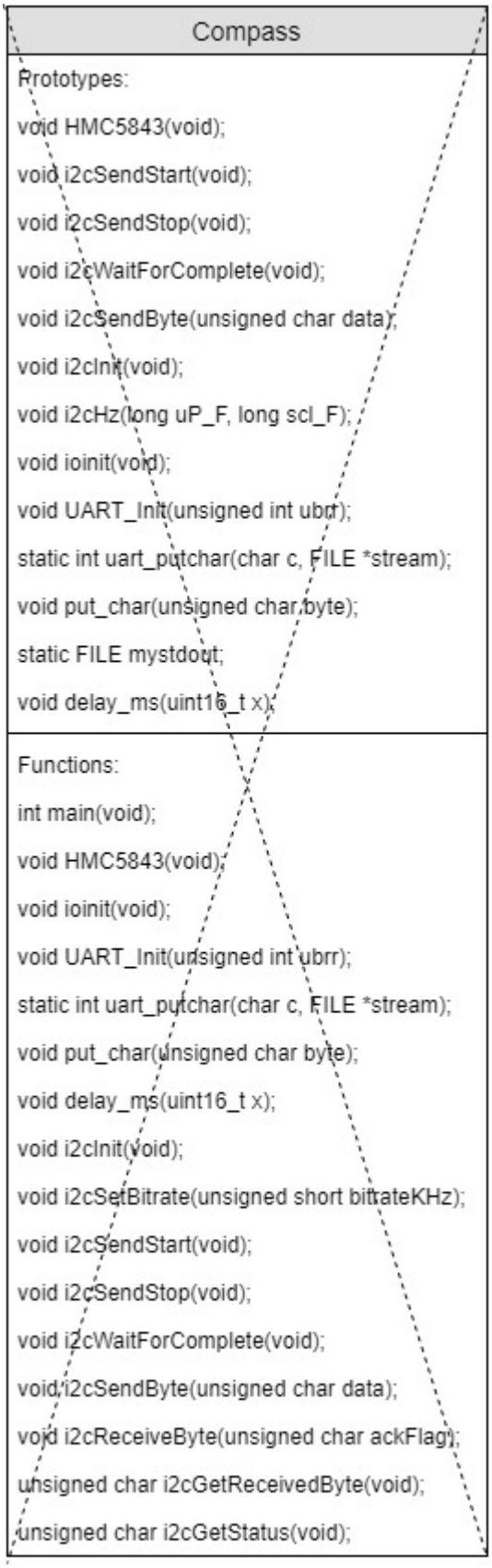
Term	*Definition*
Code conventions	These are rules that are used for programming and drawn up by the developers of Wake 'Em.
UpperCamelCase	This means that each word is capitalized. Example: SetAlarmOfAndroid.
LowerCamelCase	This means that each word is capitalized except for the first word. Example: setAlarmOfAndroid.
Odometry	The technique of measuring the amount of movement of the wheels with special sensors.
ROS	Robot Operating System, the framework Willy has been built on.

System Overview



ros_gps gps sonar odometry

Code Overview


Compass
Prototypes:
void HMC5843(void);
void i2cSendStart(void);
void i2cSendStop(void);
void i2cWaitForComplete(void);
void i2cSendByte(unsigned char data);
void i2cInit(void);
void i2cHz(long uP_F, long scl_F);
void ioinit(void);
void UART_Init(unsigned int ubrr);
static int uart_putchar(char c, FILE *stream);
void put_char(unsigned char byte);
static FILE mystdout;
void delay_ms(uint16_t x);
Functions:
int main(void);
void HMC5843(void);
void ioinit(void);
void UART_Init(unsigned int ubrr);
static int uart_putchar(char c, FILE *stream);
void put_char(unsigned char byte);
void delay_ms(uint16_t x);
void i2cInit(void);
void i2cSetBitrate(unsigned short bitrateKHz);
void i2cSendStart(void);
void i2cSendStop(void);
void i2cWaitForComplete(void);
void i2cSendByte(unsigned char data);
void i2cReceiveByte(unsigned char ackFlag);
unsigned char i2cGetReceivedByte(void);
unsigned char i2cGetStatus(void);

GPS
Prototype: void updateInfo();
Objects: TinyGPSPlus gps; SoftwareSerial ss(RXPin, TXPin);

Functions:
int main(int argc, char const *argv[]);
void updateInfo();

Sonar
Prototype: float get_measure_from(int digitalport);
ROS: ros::NodeHandle nh; sensor_msgs::LaserEcho message; ros::Publisher sonar("sonar", &message);

Functions:
void setup();
void loop();
float get_measure_from(int digitalport);

Odometry

ros::NodeHandle nh;
void messageCb(const geometry_msgs::Twist& twistMsg);
ros::Subscriber<geometry_smgs::Twist> sub("/cmd_vel", &messageCb);
geometry_msgs::Vector3 ticks;
~~ros::Publisher encoder_ticks_pub("wheel_encoder", &ticks);~~
~~Encoder LEncoder(LEncoderA, LEncoderB);~~
~~Encoder REncoder(REncoderA, REncoderB);~~

Functions:
void loop();
void setup();

Ros_gps

HMC5883L compass;
ros::NodeHandle nh;
std_msgs::String str_msg;
std_msgs::Float32 degrees;
ros::Publisher gps_pub("gps", &str_msg);
ros::Publisher compass_pub("compass", &degrees);

Prototype: void receiveEvent(int bytes);

Functions:
int main(int argc, char const *argv[]);
void receiveEvent(int bytes);

Design Decisions

In the past the decision has been made to make the hardware modular. Unfortunately the reason behind this decision is not documented.

The motor controller (odometry) is made by the group of the second semester of 2016/17. The odometry code subscribes from the topic “/cmd_vel” and publishes to the topic “wheel_encoder”. The code reads the data from the odometry sensors and publishes it to ROS, and writes data it gets from ROS to the motors.

The sonar code is reading the 10 sonar sensors and publishes it to ROS on the topic “sonar”. The code has been written by the group of the second semester of 2016/17.

The GPS and compass code is written by the group of the first semester of 2017/18. The setup is made ambiguous. The compass has a Arduino and the GPS has a Arduino. The data from the GPS is sent to the Arduino with the compass. From that Arduino the data from the Compass and the data from the GPS is both being published to two separate ROS topics. The topic with compass data is “compass” and the topic with GPS data is “gps”. This setup has not been fixed yet.

Bibliography

The current document contains no sources.