UDG_ShrimpIII user guide

Omar Alejandro Rodríguez Rosas

September 9, 2014

Contents

Chapter 1

Introduction

Chapter 2

Setup

Connect your computer to the robot using a serial cable. You can use a USB-to-serial converter if necessary.

2.1 Building with UDG_ShrimpIII

This guide uses Microsoft Visual Studio Express for Desktop 2012 and Windows 8.1 or g++ 4.7 and Ubuntu 13.10. Other compiler/OS/Distribution combnations might work as well but haven't been tested.

2.1.1 Windows

1. Copy the UDG_ShrimpIII release folder to a suitable location.
2. Open your Visual Studio solution.
3. As an optional step, you might want to add UDG_ShrimpIII include (<Release_Folder >\include) and library (<Release_Folder>\include\bin\<Architecture>) directories to your default search folders. To do so, for the include folder go to Project\<Name>Properties\Configuration Properties\C/C++\General and enter the appropriate path in Additional Include Directories. Similarly, for the binary folder edit the Additional library directories property in Project\<Name>Properties\Configuration Properties\Linker\General.
4. Your source files should use #include <UDG_ShrimpIII.h>or #include "UDG_ShrimpIII.h"[1]
5. Make sure you're linking with UDG_ShrimpIII.lib. To do so add UDG_ShrimpIII.lib to Additional Dependencies in Project\<Name>Properties\Configuration Properties\Linker\Input or add #pragma comment(lib,"UDG_ShrimpIII.lib") to your

[1] Remember that, when using quotes, If the header file is not on your current directory the full path should be included i.e. #include "c:\foo\bar\UDG_ShrimpIII.h".

code. [2]

2.1.2 Linux (Ubuntu)

1. Copy the UDG_ShrimpIII release folder to a suitable location.
2. As an optional step you might want to place the header and library files into your default search directories. Typically you'll need to copy <Release_Folder>/include/UDG_Shrimp to /usr/include/ and <Release_Folder>/bin/<Architecture>/libUDG_ShrimpIII.a to /usr/lib.[3]
3. Your source files should use #include <UDG_ShrimpIII.h>or #include "UDG_ShrimpIII.h"[4]
4. To build your application use -std=c++11 [5] and link with -lUDG_ShrimpIII.a [6]

[2] Use the whole path if you skipped step 3.

[3] These default libraries might vary depending on your compiler settings.

[4] Remember that, when using quotes, If the header file is not on your current directory the full path should be included or you can also use the -I option (see g++ help).

[5] -std=c++0x might work as well if you're using an older g++ version

[6] Or simply UDG_ShrimpIII.a if you skipped step 2. If the file is not on your current folder you can use its whole path or the -L option (see g++ help).

Chapter 3

UDG_ShrimpIII API

3.1 Data types and definitions

3.1.1 #define S_ERROR_UNKNOWN_COMMAND 0x80

Error code that indicates an unknown command has been received. This is most likely a communication error.

3.1.2 #define S_ERROR_ARGUMENT 0x81

Indicates that a wrong argument has been received by the robot.

3.1.3 #define S_ERROR_I2C 0x82

Indicates an I^2C communication error.

3.1.4 #define S_ERROR_LIMIT_REACHED 0x83

Indicates that the upper limit for a given parameter has been reached.

3.1.5 #define RAW_BYTE unsigned char

Unsigned byte-wide data type.

3.1.6 #define SIGNED_BYTE char

Signed byte-wide data type

3.1.7 #define S_BUFFER_SIZE 1024

The default buffer size in bytes for the communication.

3.1.8 #define UNSIGNED_INTEGER32 unsigned int

Unsigned 32-bit integer.

3.1.9 #define UNSIGNED_INTEGER16 unsigned short

Unsigned 16 bit integer.

3.1.10 struct FirmwareVersion

```
struct FirmwareVersion
{
        RAW_BYTE major;
        RAW_BYTE minor;
        RAW_BYTE patch;
};
```

A stuct to contain version information about the robot.

3.1.11 struct WheelEncoders

```
struct WheelEncoders{
        UNSIGNED_INTEGER32 F;
        UNSIGNED_INTEGER32 FL;
        UNSIGNED_INTEGER32 FR;
        UNSIGNED_INTEGER32 BL;
        UNSIGNED_INTEGER32 BR;
        UNSIGNED_INTEGER32 B;
};
```

Value F: current value of the front wheel encoder *Value FL:* current value of the front left wheel encoder *Value FR:* current value of the front right wheel encoder *Value BL:* current value of the rear left wheel encoder *Value BR:* current value of the rear right wheel encoder *Value B:* current value of the rear wheel encoder

Contains information from the robot's wheel encoders

3.1.12 struct RobotStatus

```
struct RobotStatus
{
        unsigned char ROB_ON :1;
        unsigned char ROB_STOPPED :1;
        unsigned char IR_ENABLED :1;
        unsigned char UNUSED;
};
```

ROB_ON: Robot is switched on *ROB_STOPPED:* Robot is stopped by an emergency stop *IR_ENABLED:* Infrared remote control is enabled *Bits 3 - 7:* Unused

Contains general informatiomn about the robot status.

3.1.13 struct PowerSupplyStatus

```
struct PowerSupplyStatus
{
        RAW_BYTE ALL_OK  :1;
        RAW_BYTE IN_LOW  :1;
        RAW_BYTE VIN_MIN  :1;
        RAW_BYTE VIN_SECURE:1;
        RAW_BYTE VIN_ERROR  :1;
        RAW_BYTE VIN_HI  :1;
        RAW_BYTE unused  :1;
        RAW_BYTE D2_OVER;
};
```

ALL_OK: Power supply is ok, i.e. none of the bits below are set *IN_LOW:* Battery voltage is below first security threshold Vlow *VIN_MIN:* Battery voltage is below second security threshold Vmin, power to motors and servos will be switched off *VIN_SECURE:* Battery voltage is below third security threshold Vsecure, power supply will shut down *VIN_ERROR:* Battery voltage is low and 2nd derivative is too high, power supply will shut down *VIN_HI:* Battery voltage is above Vmax, power supply will shut down *Bit 6:* Unused *D2_OVER:* 2nd derivative of battery voltage has been too high for a long time, power supply will shut down

Contains general information about the robot's battery status.

SrvoAndMotorCommands

```
struct SrvoAndMotorCommands
{
        UNSIGNED_INTEGER16 servoF;
        UNSIGNED_INTEGER16 servoB;
        UNSIGNED_INTEGER32 motorF;
        UNSIGNED_INTEGER32 motorL;
        UNSIGNED_INTEGER32 motorR;
        UNSIGNED_INTEGER32 motorB;
};
```

servoF: command to the front servo *servoB:* command to the rear servo *motorF:* command to the front motor *motorL:* command to the left motors *motorR:* command to the right motors *motorB:* command to the rear motor

structDigitalInputs

```
struct DigitalInputs
{
        RAW_BYTE reserved0  :1;
```

```
        RAW_BYTE nESTOP :1;
        RAW_BYTE GPIO :1;
        RAW_BYTE reserved3;
};
```

Bit 0: Reserved *nESTOP:* State of active-low emergency stop input *GPIO:* State of general-purpose input *Bit 3 - 7:* Reserved

Contains the status of the digital inputs

3.1.14 structRC5Frame

```
struct RC5Frame
{
        RAW_BYTE address;
        RAW_BYTE data;
};
```

Contains information to communicate through RC5;

3.2 UDG_ShrimpIII class

3.2.1 Public members

Most of these fields update themselves when the appropriate function is called.

FirmwareVersion firmwareVersion A FirmwareVersion struct to contain information about the firmware. Get updated when calling ReadFirmwareVersion().

SIGNED_BYTE velocity

velocity at which the robot travels. This field gets updated when calling SetVelocityAndSteeringAngle() or ReadVelocityAndSteeringAngle(). The value might be innaccurate if it's not updated often.

SIGNED_BYTE steeringAngle

Steering angle of the robot. This field gets updated when calling SetVelocityAndSteeringAngle() or ReadVelocityAndSteeringAngle(). The value might be innaccurate if it's not updated often.

WheelEncoders wheelEncoders

Contains information about the robot's wheel encoders. This field gets updated when calling ReadWheelEncoderValues(). The value might be innaccurate if

it's not updated often.

RobotStatus robotStatus

Contains information about the robot's general status. This field gets updated when calling ReadRobotStatus().The value might be innaccurate if it's not updated often.

PowerSupplyStatus powerSupplyStatus

Contains information about the robot's power supply. This field gets updated when calling ReadPowerSupplyStatus().The value might be innaccurate if it's not updated often.

float voltage

Represents the battery voltage in volts. This field gets updated when calling ReadBatteryVoltage().The value might be innaccurate if it's not updated often.

SrvoAndMotorCommands servoAndMotorCommands

Contains information about the robot's motors. This field gets updated when calling WriteLowLevelServoAndMotorCommands() or ReadLowLevelServoAndMotorCommands.The value might be innaccurate if it's not updated often.

3.2.2 DigitalInputs digitalInputs

Contains information about the robot's digital inputs. This field gets updated when calling ReadDigitalInputs().The value might be innaccurate if it's not updated often.

RAW_BYTE shrimpBuffer[S_BUFFER_SIZE]

This is the raw byte buffer for the robot. It should be used for debug purposes only.

UDG_ShrimpIII(const char* serialPortName)

Constructor method for the class. Receives a null terminated string representing the name of the associated serial port ("COMX on Windows or "/dev/ttyX"

on Linux).

UDG_ShrimpIII()

Default destructor for the class.

bool NoOperation()

This command doesn't have any effect. It can be used as a "ping" to ensure that communication with the robot is working. It can also be used in case of loss of synchronization in the command stream: sending a string of 0x00 bytes at least as long as the longest command available ensures that the command following the 0x00 bytes will be interpreted correctly.
Return value: True if the function succeeds, false if it doesn't.

bool ReadFirmwareVersion()

This command stores the version of the controller's firmware in firmwareVersion. Buffer receives —0x01— Major (8) —Minor (8) —Patch (8)—
For example, for version 1.4.7, Major=1, Minor=4 and Patch=7.
Return value: True if the function succeeds, false if it doesn't.

bool TurnRobotOn()

his command connects power to the motor and servo controllers, enabling the movement of the robot.
Return value: True if the function succeeds, false if it doesn't.

bool TurnRobotOff()

This command removes power from the motor and servo controllers. Power consumption goes down to a minimum, so it can be used to conserve power when the robot is not moving.
Return value: True if the function succeeds, false if it doesn't.

bool SetVelocityAndSteeringAngle(SIGNED_BYTE velocity, SIGNED_BYTE angle)

This command sets new target values for the linear velocity and steering angle of the robot. The motor and servo controllers will be reprogrammed immediately to reach the new targets.

Return value: True if the function succeeds, false if it doesn't.

Velocity (-127..127): speed of the rear wheel. The value -127 is full speed backward, 127 is full speed forward, and 0 is stopped.
angle (-90..90): angle of the rear wheel relative to the straight line position, in degrees. Negative values are angles to the right (meaning that the robot will turn to the left), positive values to the left.
Updates velocity and steering angle members.
Return value: True if the function succeeds, false if it doesn't.

bool ReadVelocityAndSteeringAngle()

This command updates velocity and steeringAngle of the robot, and can be used for example to log the commands that are sent to the robot with the infrared remote control.
Buffer receives —0x05— Velocity (8)— Steering angle (8) —
Return value: True if the function succeeds, false if it doesn't.

bool EmergencyStop()

This command stops all motors on the robot (i.e. sets their speeds to zero). The steering servo positions remain unchanged.
Return value: True if the function succeeds, false if it doesn't.

bool ReadWheelEncoderValues(WheelEncoders* encoders)

This command reads and returns the encoder values of all six wheels, allowing to make simple odometry calculations or to get a feeling of how far the robot has travelled. The wheelEncoders property will be updated. See struct WheelEncoders definition for more details.
WheelEncoders encoders:* pointer to a struct used as a return value, this is a copy of the value at wheelEncoders property. Set to null if its not important to you.
Buffer Receives— 0x07— Value F (32) —Value FL (32) —Value FR (32)— Value BL (32)— Value BR (32)— Value B (32) —
Return value: True if the function succeeds, false if it doesn't.

bool ReadRobotStatus(RobotStatus* status)

This command returns the current status of the robot. Updates the robotStatus property and, if provided, stores the same value at status, a pointer to a RobotStatus struct.

Return value: True if the function succeeds, false if it doesn't.

float ReadBatteryVoltage()

This command reads the current voltage of the battery on the robot. It is updated about once per second, so reading it more often will return the same value between updates. Automatically updates voltage property. **Return value:** The voltage as a float value. Returns 0 if an error occurs.
Buffer receives — 0x09— Battery voltage (8) —
NOTE: The battery voltage received at the buffer must be multiplied by 0.0625f to obtain the actual value. Both the return value and the property update autmatically perfrom this multiplication.

bool ReadPowerSupplyStatus(PowerSupplyStatus* status)

This command returns the current status of the power supply controller. It reports any problems the power supply controller could identify, for example low battery states. The security thresholds satisfy the following condition:
$V_{min} < V_{secure} < V_{low} < V_{max}$
Updates the powerSupplyStatus property and, if provided, stores the same value at status, a pointer to a powerSupplyStatus struct.
Return value: True if the function succeeds, false if it doesn't.

bool DisableInfraredRemoteControl()

This command disables the infrared remote control decoder. It can be useful if the robot is controlled through RS-232 and there are infrared transmitters in the vincinity.
Return value: True if the function succeeds, false if it doesn't.

bool EnableInfraredRemoteControl()

This command enables the infrared remote control decoder.
Return value: True if the function succeeds, false if it doesn't.

bool MutePowerSupplyBuzzer()

On power-up, the power supply controller is programmed to make various sounds when the battery level goes below certain thresholds. This command mutes those sounds.
Return value: True if the function succeeds, false if it doesn't.

bool UnmutePowerSupplyBuzzer()

This command enables batter-level sounds of the power supply controller. This is the default power-up state.
Return value: True if the function succeeds, false if it doesn't.

bool WriteAn8BitRegisterOnAnI2CModule(RAW_BYTE moduleAddress, RAW_BYTE registerAddress, RAW_BYTE value)

This command writes an 8-bit register on an I2C module. The master acts as a pass-through.
Return value: True if the function succeeds, false if it doesn't.

bool ReadAn8BitRegisterOnAnI2CModule(RAW_BYTE moduleAddress, RAW_BYTE registerAddress, RAW_BYTE* value)

This command reads an 8-bit register from an I2C module. The master acts as a pass-through. The value is stored at value, a pointer to a RAW_BYTE variable.
Return value: True if the function succeeds, false if it doesn't.

bool WriteA32BitRegisterOnAnI2CModule(RAW_BYTE moduleAddress, RAW_BYTE registerAddress, UNSIGNED_INTEGER32 value)

This command writes a 32-bit register on an I2C module. The master acts as a pass-through.
Return value: True if the function succeeds, false if it doesn't.

bool ReadA32BitRegisterOnAnI2CModule(RAW_BYTE moduleAddress, RAW_BYTE registerAddress, UNSIGNED_INTEGER32* value)

This command reads a 32-bit register from an I2C module. The master acts as a pass-through. The value is stored at value, a pointer to a UNSIGNED_INTEGER32 variable.
Return value: True if the function succeeds, false if it doesn't.

bool ReadLowLevelServoAndMotorCommands(SrvoAndMotorCommands* motors)

This command reads and returns the low-level commands sent to the servos and motors. It should only be used as a debugging aid.
Buffer receives —0x13 —Servo F (16)— Servo B (16)— Motor F (32)— Motor

L (32)— Motor R (32)— Motor B (32)—

bool SystemReset()

This command resets the main controller and re-initializes the complete system.
Return value: True if the function succeeds, false if it doesn't.

bool ReadLastRC5Frame(RC5Frame* frame)

NO INFO

bool ReadDigitalInputs(DigitalInputs* inputs)

This command returns the current state of the digital inputs to the microcontroller. The emergency stop input is active-low, i.e. the robot is stopped when nESTOP=0.
Return value: True if the function succeeds, false if it doesn't.

bool WriteLowLevelServoAndMotorCommands(SrvoAndMotorCommands motors)

This command writes low-level commands directly to the servos and motors, bypassing the robot model. No checking is done on the values, such as out-of-bounds conditions, so it might be possible to damage the hardware by writing bad values. This command should normally not be used.
Return value: True if the function succeeds, false if it doesn't.

3.3 Private members

bool ExecuteInstruction(RAW_BYTE* response,int numberOfInstructionBytes,int numberOfResponseBytes,const char* name)

Reads/Writes instructions from/to the serial port.
Return value: True if the function succeeds, false if it doesn't.

Chapter 4

Code sample

The following source code (for both linux and windows) demonstrates some of the common operations the robot can perform. Notice how some functions return a value, others update the robot's properties and some can do both using an external pointer that can be ignored by setting it to NULL.

```
#include "UDG_ShrimpIII.h"
#include <stdio.h>
#ifdef __linux
#include <unistd.h>
#else
#include <Windows.h>
#endif
int main()
{
        DigitalInputs dInputs;
        WheelEncoders wEncoders;
        //intialization
        UDG_ShrimpIII robot("COM3");
        //Start receiving serial commands
        robot.TurnRobotOn();
        //ping the robot. Ignoring return value
        robot.NoOperation();
        //update UDG_ShrimpIII properties directly
        if(robot.ReadBatteryVoltage())
                printf("voltage: %fv\n",robot.voltage);
        if(robot.ReadFirmwareVersion())
                printf("Firware version: %i.%i.%i\n",robot.firmwareVersion.major,
                robot.firmwareVersion.minor,robot.firmwareVersion.patch);
        //Update properties and store reading to a variable to be used later
        if(!robot.ReadDigitalInputs(&dInputs))
```

```
                printf("error reading digital inputs\n");
        if(!robot.ReadWheelEncoderValues(&wEncoders))
                printf("Error reading wheel encoders\n");
        //Update properties without saving value to a variable
        if(!robot.ReadRobotStatus(NULL))
                printf("Error reading robot status\n");
        //use previously stored values
        printf("Digital inputs: GPIO = %X        nESTOP = %X\n",dInputs.GP
        printf("Wheel encoders: B = %u  BL = %u BR = %u F = %u  FL = %u F
        //move in a staight line for 2 seconds
        robot.SetVelocityAndSteeringAngle(20,0);
#ifdef __linux
sleep(2)
#else
        Sleep(2000);

#endif
        robot.SetVelocityAndSteeringAngle(0,0);


        robot.TurnRobotOff();

        return 0;
}
```