Simple Communication Protocol for Rapid Robot Prototyping

Roland Stelzer and Karim Jafarmadar

***Abstract*— Aim of the method presented in this paper is to provide a simple and unique communication protocol for a wide variety of sensors and actuators. All of the devices are equipped with a microcontroller in order to translate the proprietary device dependent protocol into the protocol presented here, named Simple Sensor Network (SSN). Due to the very simple commands to request data from a sensor respectively to send data to an actuator via RS232, it is an appropriate way for a novice to get in touch with robotics without the need of expert knowledge in electronics or programming. SSN makes a system easy to maintain, to adapt, and to extend.**

I. Introduction

Usually an autonomous robotic system consists of sensors, actuators and a computer controlling both device types and applying algorithms to them. Similar sensors and actuators often provide totally different interfaces. Therefore reimplementation and testing of interfaces requires a substantial part of development time. The proposed system provides a simple and unique communication protocol for a wide variety of sensors and actuators. The protocol is named Simple Sensor Network (SSN). A sensor or actuator in combination with a microcontroller forms an SSN module. Purpose of the microcontroller is to translate the proprietary device dependent protocol into SSN commands.

Once an SSN controller is developed for a certain sensor or actuator it can be reused in other applications by “plug and play”. The device network is scaleable in terms of numbers of devices and every device can be addressed individually. In contrast to common protocols used on embedded systems like I²C [1] or CAN [2] the proposed system is based on RS232 standard which makes it easily connectable to both personal computers and microcontrollers.

A light following SSN robot demonstrates the practicability and simplicity of the protocol. Furthermore SSN is implemented and tested successfully on the "Roboat" [3,4], a fully autonomous sailboat which won the Microtransat [5,6,7], an international competition in autonomous sailing.

II. Hardware Topology

An SSN topology consists of at least one SSN module – sensor or actuator – and a master. The master controls the entire system. The communication is always initiated by the master. An SSN module can directly be connected to the master’s serial port. In this case the number of SSN modules is limited to the number of serial ports of the master. If a PC is used as master system, usually not more than one or two serial interfaces are available.

To connect more SSN modules an additional device, the SSN switch, can be put in between the master and the SSN modules (Fig. 1). Up to 15 SSN modules can be connected to a single SSN switch. The SSN switch itself is connected to the master and listens for a request packet coming from the master. The SSN packet includes an address from 0x0 to 0xE, which specifies the port of the SSN switch to which the desired SSN Module is connected.

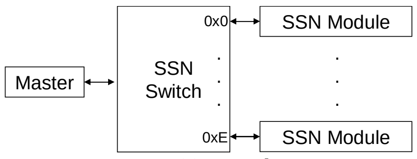

Fig. 1: SSN Topology

For systems with more than 15 SSN modules, SSN switches can be cascaded (Fig. 4, Fig. 5).

III. Communication Flow

The communication between master, SSN switch and SSN modules is always initiated by the master. The master can either set an actuators value or gather a sensor value. In both cases a request packet has to be sent from master to SSN switch. When a request occurs the SSN switch acts as follows:

1. SSN switch receives request packet from master.
2. SSN switch extracts the address part of the packet.
3. SSN switch forwards the complete packet to the switch-port corresponding to the address extracted before.
4. SSN switch waits for a reply packet from the SSN module and routes it back to the master. If no reply occurs within 100 ms, a timeout message will be returned to the master by the SSN switch.

A. *Request Packet*

The Request Packet consists of a header byte and an optional data part. The header is divided into four bits for the module address and four bits for the command, which has to be applied to the particular SSN module.

Roland Stelzer and Karim Jafarmadar are with the Austrian Association for Innovative Computer Science, Kampstraße 15/1, 1200 Vienna, Austria. E-Mail: roland.stelzer@innoc.at and karim.jafarmadar@innoc.at

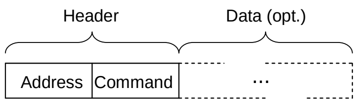

Fig. 2: Request Packet

Valid addresses are numbers from 0x0 to 0xE. 0xF indicates cascading of multiple SSN switches. The commands can be classified into three groups (Table I):

- Get general information about a particular SSN module (command id 0x0 – 0x1).
- Get value of a sensor device respectively status of an actuator (command id 0x2 – 0x5).
- Set value of an actuator device respectively change settings of a sensor (command id 0x6 – 0x9). These commands contain a data part in addition to the header. The length of data depends on the command.

TABLE I
SSN COMMANDS

Command ID	Command Description	Data length (bytes) request / reply
0x0	getinfo short	0 / 1
0x1	getinfo long (0-terminated)	0 / variable
0x2	getdata string (0-terminated)	0 / variable
0x3	getdata byte	0 / 1
0x4	getdata word	0 / 2
0x5	getdata double word	0 / 4
0x6	setdata string (0-terminated)	variable / 0
0x7	setdata byte	1 / 0
0x8	setdata word	2 / 0
0x9	setdata double word	4 / 0
0xA – 0xE	<not in use>	-
0xF	error indicator in reply packets	-

The commands *getinfo short* and *getinfo long* are supported by every SSN module. The reply packet on a *getinfo short* request gives information about the additionally supported commands. *getinfo long* returns a textual description of the particular SSN module.

B. Standard Reply Packet

When an SSN module receives a request packet, it processes the command and returns a reply packet. The first byte of the reply packet is identical to the first byte of the preceding request. Thus the most-significant half-byte contains the address (SSN switch port number where the module is connected to) and the least-significant half-byte describes the command. By means of the command, the SSN switch determines the number of bytes following the header byte (Table I).

C. Info Reply Packet

The commands *getinfo short* and *getinfo long* deliver general information about an SSN module. *getinfo long* returns a textual description of the SSN module as a 0-terminated string following the unmodified request header.

The reply packet on a *getinfo short* request gives information about the additionally supported commands.

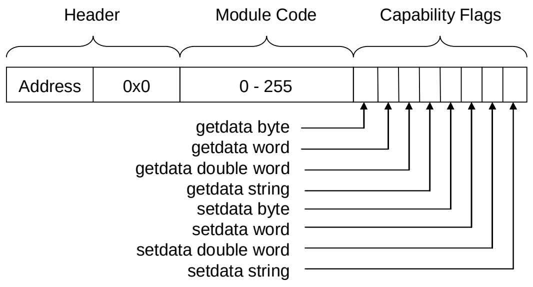

Fig. 3: Reply Packet for getinfo short

In addition to the unmodified request header the reply packet contains a module code, which can be used by the module developer to categorize SSN modules. A third byte informs about the range of implemented SSN commands (Fig. 3).

D. Timeout Reply Packet

If the SSN switch does not receive a reply packet from the SSN module within 100 ms a timeout occurs. In this case the SSN switch generates a timeout reply packet which consists of the address of the particular SSN module and the error indicator 0xF. No data bytes follow. A reply packet after timeout will be ignored.

IV. CASCADING

Through cascading of multiple SSN switches, systems with more than 15 SSN modules can be set up (Fig. 4). Each SSN switch can be equipped with a special cascade port with the cascade address 0xF.

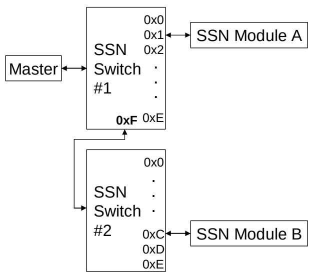

Fig. 4: Cascade Port

If more than one SSN switch has to be passed on the way between master and SSN Module, an extra header byte is necessary for each additional SSN switch. The most-significant half-byte of this header contains 0xF. This indicates that it is a cascade-header and the direct recipient of the packet is not an SSN module, but an additional SSN switch. The least-significant half-byte of the cascading header contains the port number to which the additional SSN switch is connected.

If the SSN switch receives a cascade header it discards the first byte. It forwards the rest of the packet to the following SSN switch and waits for a reply like with directly connected SSN modules. With this strategy, any number of SSN switches can be cascaded, as long as the packet runtime does not exceed the timeout limit.

Not only the standard cascade port 0xF, but any of the ports can be used to cascade (Fig. 5). By using multiple ports for cascading the number of hops between master and SSN module and the packet runtime can be minimized.

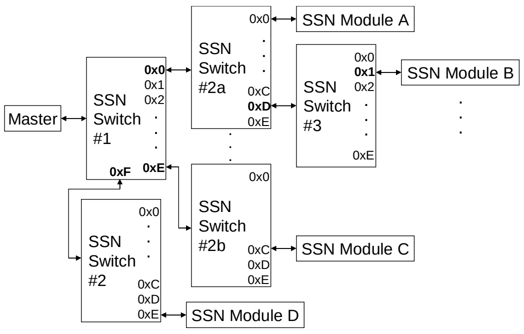

Fig. 5: Complex SSN Topology

V. Communication Example

The demonstration Topology is illustrated in Figure 5. SSN Module B is supposed to be a distance sensor. The command *getdata byte* (0x3) requests the measured distance in cm as 8 bit integer. The data flow from master to module is shown in Figure 6.

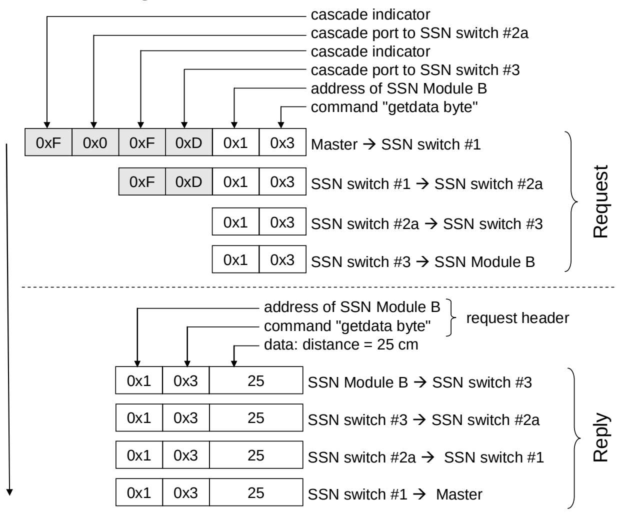

Fig. 6: Communication Example acc. to Fig. 5 - getdata byte from SSN Module B

VI. Application Example

For experiments controller hardware of SSN modules is based on Microchip PIC microcontrollers [8]. Software templates programmed in C [9] allow easy and fast development of new SSN modules.

A. *Light Following SSN Robot*

The robot in Figure 7 consists two side mounted independently driven wheels which are used for both propulsion and steering. This robot has a caster wheel mounted at the front to keep it from falling over. The robot is equipped with two brightness sensors, one pointed to the left, the other one to the right. The brightness sensors as well as the motor controllers for the wheels are connected to an SSN switch. Aim of the robot's algorithm is to follow a light source. The algorithm is executed by the master, which can be a laptop computer.

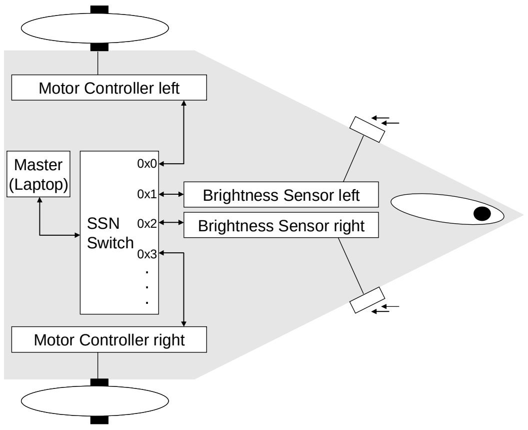

Fig. 7: Light Following SSN Robot

Only the SSN modules know about the device dependent protocols of the sensors and actuators and translate them to SSN commands. The master knows the SSN switch port number of every connected SSN module. The master communicates with the SSN modules using simple SSN commands (*setdata byte*, *getdata byte*). Therefore a few lines of code suffice to implement a light following robot.

Pseudo code example:

```
left=getdata_byte(0x1)    // read brightness left
right=getdata_byte(0x2)   // read brightness right
if ( left > right )       // light is left -> go to left
  setdata_byte(0x0,100)   // set left motor speed to 100
  setdata_byte(0x3,200)   // set right motor speed to 200
else                      // light is right -> go to right
  setdata_byte(0x0,200)   // set left motor speed to 200
  setdata_byte(0x3,100)   // set right motor speed to 100
```

B. *Autonomous Sailboat "Roboat"*

SSN is successfully implemented and used on the autonomous sailboat "Roboat". The boat won in the first Microtransat Challenge for autonomous sailboats in Toulouse, France, June 2006. The "Roboat" demonstrated completely autonomous sailing, where routeing, navigation and carrying out the manoeuvres run automatically and directly on the boat. For this purpose a sailboat is equipped with various sensors to measure the environmental conditions and actuators to control the rudder and sails. These devices are connected to the master via an SSN switch and communicate through SSN (Fig. 8).

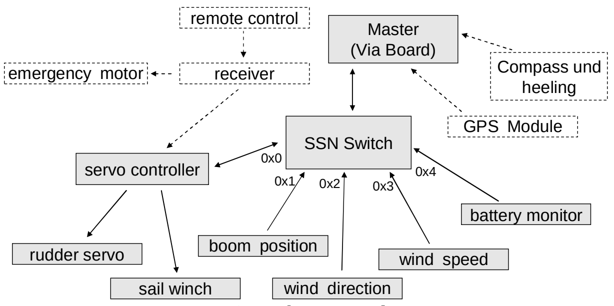

Fig. 8: "Roboat" Topology

VII. CONCLUSIONS

A robotic system applies an algorithm to a set of sensors and actuators. Different sensors respectively actuators usually mean different ways of communication. SSN provides a set of commands which allow uniform communication to a wide range of devices. Due to its simple command structure SSN is a proper way for novices to set up a robotic application. Once an SSN module is available there is no need to care about the specific characteristics of the sensor/actuator hardware.

The protocol is based on standardized serial communication (RS232). Therefore the SSN modules can be connected to a lot of different systems (PC, PDA, microcontrollers, etc.) and all popular programming languages can be used. Up to 15 SSN modules can be connected to a single SSN switch. Cascaded SSN switches allow larger installations.

The standardized set of SSN commands allows replacing sensors/actuators without changes in the master's program code. As an example, it is possible to replace an infrared distance sensor with an ultrasonic one.

SSN is successfully implemented and used on the autonomous sailboat "Roboat" and shows its strength especially in the prototyping phase where the system topology and the used devices frequently change. SSN makes a robot prototype easy to set up, to adapt, and to extend.

REFERENCES

[1] Philips Semiconductors, "The I²C-Bus Specifications", Version 2.1, Jan. 2000 [online]. Available: http://www.nxp.com/acrobat_download/literature/9398/39340011.pdf

[2] J.A. Gil, A. Pont, G. Benet, F.J. Blanes , M.Martínez "A CAN Architecture for an Intelligent Mobile Robot". *Proceedings of IFAC Symposium on Intelligent Components and Instruments for Control Applications SICICA'97*, pp.65-70. Annecy, 1997.

[3] R. Stelzer, T. Pröll, and R.I. John, Fuzzy Logic Control System for Autonomous Sailboats, Accepted for publication at *IEEE International Conference on Fuzzy Systems 2007*, London, UK.

[4] R. Stelzer and T. Pröll, "Autonomous sailboat navigation for short course racing," submitted for publication, 2007.

[5] Y. Briere, "First Microtransat Challenge" [online]. Available: www.ensica.fr/microtransat, 2006.

[6] C. Sauze and M. Neal, "An Autonomous Sailing Robot for Ocean Observation," in *Proceedings of TAROS Conference 2006* [online]. Available: http://taros.mech.surrey.ac.uk/papers/Sauze_Neal.pdf, 2006.

[7] M. Neal, "A hardware proof of concept of a sailing robot for ocean observation," *IEEE Journal of Ocean Engineering*, vol. 31, pp. 462-469, 2006.

[8] Microchip, "PIC Microcontrollers" [online]. Available: http://www.microchip.com

[9] Custom Computer Services Inc., "CCS C Compiler for PIC Microcontroller", [online]. Available: http://www.ccsinfo.com