Working paper
2005:04

Henrik Nilsson

GEOMETRY MEASUREMENT USING M-SPOT 90

UNIVERSITY
TROLLHÄTTAN • UDDEVALLA

Department of Technology, Mathematics & Computer Science

Research report

Geometry measurement using M-Spot 90

Henrik Nilsson

Summary

The work presented in this report is conducted within the research project Metal Deposition in the Mecatronics for Industrial Applications group at the University of Trollhättan/Uddevalla.

The main aim of this report is to present a LABView program that have the ability to extract geometry information from a laser sensor.

The project was conducted in three steps where the first one was to learn about the communication protocol of the sensor, the second step was to develop a C-program with the ability to communicate with the sensor and the last step was to develop the LABView program with the ability to extract geometry information from the sensor.

From the sensor manual it was learned that the sensor transmits 1034 bytes for the geometry. All of the bytes are not necessary to make a diagram of the measured profile.

The aim of the report have been accomplished in the work but there are still some issues left before the sensor is ready for use in automated robot metal deposition. Two of the must important are how long the sensor can be borrowed from Lund and what specific area the sensor will be used to measure on. These two have to be decided before any further work is carried out with the sensor.

Contents

1 Introduction

One of the research groups active at the University of Trollhättan/Uddevalla (HTU), Sweden, is the MIA-group (Mechatronics in Industrial Applications), which aims for research on controlling and supervision of industrial processes. The group was founded during the winter of 2003/2004 as a parallel and co-operating research group to the successful research group VIP (Verkstadsindustriprocesser), also at HTU.

At this point a couple of different projects have been undertaken by the research group among them the development of a sensor and control system for automated robot metal deposition using welding technique. Within this project many smaller areas are researched as starting points of the project, among them a control interface for a suitable geometry sensor.

1.1 Background

Metal deposition is a method where a metal feature is built by adding welding material to create the wanted shape. A common way to build a part is to add material by powder or wire. The material is melted using a heat source, e.g. a tungsten electrode (TIG welding equipment) or laser. The idea of the method is to building up layers to created the wanted products or add metal features, like flanges, bosses and pads, to existing products.

Some areas of interest for metal deposition are to build prototypes, adding features to parts and repairing damaged parts. The advantage of using metal deposition for building prototypes compared to the already available rapid prototyping (RP) and free form fabrication (FFF) methods is that the created part is in metal and therefore has the material properties of the finished part. This makes the method suitable for one of a kind production or small series. The ability to add features to a part gives possibilities to improve and import changes to old parts and also the possibility to use a simple and low cost base feature and adding advanced features by using metal deposition to generate a complex part.

The possibility to repair damage parts instead of discarding them can make a huge saving if the part is expensive, e.g. cast of titanium or stainless steel or tools.

To be able to automate the metal deposition process the use of sensors are essential. One of the most important things to measure is the geometry before the welding equipment to determine the appropriate distance to the weld and the amount of welding material to deposit to achieve a surface with the programmed geometry.

For the work presented in this report a certain sensor, a M-Spot 90 from ServoRobot was used. It was designed to be used as a seam tracker using laser technology.

1.2 Aim and objectives

The main aim of the project was to develop a method that collects the measured information from the M-Spot 90 and transforms it into a useful format for a control system.

The objectives of this project were:

- Design a program that collects the measured information from M-Spot 90
- Design an algorithm that transfers the collected information into usable data

1.3 Restrictions

The complete control system will not be developed in this report.

The programs developed are designed for use together with the chosen sensor.

1.4 Methodology

The approach for developing the sensor control program was conducted in three steps, researching the protocol of the sensor, develop a simple C-program to make sure the communication works as it should and finally develop a LABView based geometry measuring system for the sensor.

2 Sensor

The sensor used is M-Spot 90. The two main reasons for using that sensor are that it is made for use in a welding environment and that it can be borrowed from Lund Institute of Technology University. The M-Spot 90's primary area of use is as a seam tracker for robot welding applications and is of the brand ServoRobot.

2.1 M-Spot 90

The M-Spot 90 is a laser line scanner with 256 scanning points in a straight line.

The M-Spot 90 has a user interface, a PC software called USER2000. The software is used to setup the sensor parameters for the suitable use in the current application. The communication between the sensor PC port and the user PC is conducted via a RC232 serial link. see Figure 1.

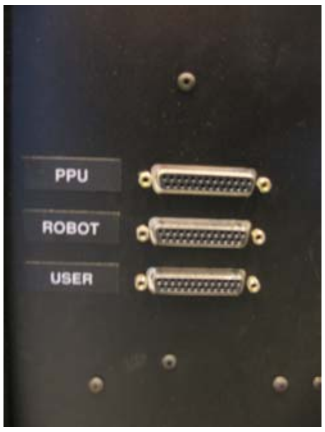

Figure 1 – Shows the serial ports on the sensor. For connection to a PC and the USER2000 interface use the USER port, for use with the developed interface use the ROBOT port.

In USER2000 a profile of the measurement from the sensor can be displayed. The update time varies between 2 and 5 seconds.

2.2 Protocol

The protocol used to control the sensor can be found in the manual for the sensor[1].

In the manual the command to send to the sensor to request all the values from the sensor is {STX}{Z}{G}{DLE}{ETX}{CKS}. The sensor will reply with a message in the following form:

{STX}{z}{status}{t1}{t2}{t3}{t4}{ym0}{yl0}…{ym255}{yl255}

{zm0}{zl0}…{zm255}{zl255}{DLE}{ETX}{CKS}

The commands in the sequences have the following meaning:

{STX} – Start of text, decimal number 02

{Z}{G} – The command to the sensor for requesting the profile, decimal numbers 90 and 71

{DLE} – Sending command, decimal number 16

{ETX} - End of text, decimal number 3

{CKS} – Checksum, calculated from the message using xor-function

{z} – Answer from sensor that a Z-command was received, decimal number 122

{status} – Indicates that the correction is normal (‘c’) or that no data is available (‘z’)

{t1, t2, t3, t4} – Indicates the time in milliseconds, t1 is the most significant, t4 the least

{ym0, yl0…ym255, yl255} – Profile values in y-direction for the measurement, m indicates most significant bit and l least significant bit

{zm0, zl0…zm255, zl255} - Profile values in z-direction for the measurement, m indicates most significant bit and l least significant bit

Each of the measured values are transmitted in two parts a most significant bit, indicated by the m (ym0) and a least significant bit indicated by the l (yl0). To calculate the value for one pixel the formula is:

256*ym0+yl0

One problem that is encountered when the measured value 16 is sent is that 16 also is the decimal value for the command DLE. DLE is used to tell the receiving unit that the next byte sent is not a measured value but a command. Because not all the sent 16 in a transmission is preceding a command the sensor has a special way to send the measured value 16 and it is to send the 16 twice.

Checksum (CKS) is calculated from the transmission excluding STX, DLE, ETX and CKS using xor with binary forms of the numbers. For the request profile command (a Z followed by a G) the checksum value will be 29, see Table 1 for details.

	Decimal	**Binary**
Z	90	01011010
G	71	01000111
CKS	29	00011101

Table 1 – Xor calculation of the request profile command Z followed G.

This makes the message to send to the sensor to request profile:

{2}{90}{71}{16}{3}{29}

The total amount of bytes to receive from the sensor is 1034 with 7 start of transmission bytes, 512 bytes with y values, 512 bytes with z values and 3 end of transmission bytes.

3 C program

A C program was developed investigate protocol information from the manual. The program was developed together with Dr Fredrik Danielsson and Mr Mattias Ottosson. The program can be seen in Appendix xxx. In the program the sending of double 16 for measured values is compensated and each of the received bytes printed and the values for each of the pixels is calculated and displayed.

Even tough the program does receives all of the bytes, somewhere in the program there is a type conversion that makes the highest value of the bytes to be 127 and not 255 as required. An example:

When the measured value {49}(msb, most significant byte) {127}(lsb, least significant byte} increases with 2 the program will print {49}{1}. If the measured value continues to increase it will print {49}{127}again. If the value increases 2 more the result will be {50}{1}. The problem causing this error should not be hard to find but the program still fulfils its purpose.

4 LABView program

With starting point from the developed C program a LABView program was designed with assistance from Mr Mattias Ottosson at HTU. The program contains five parts, open serial port, send message request profile, receive profile, close serial port and calculate z.

4.1 Open serial port

The part of the LABView program that opens the serial port can be seen in Figure 2.

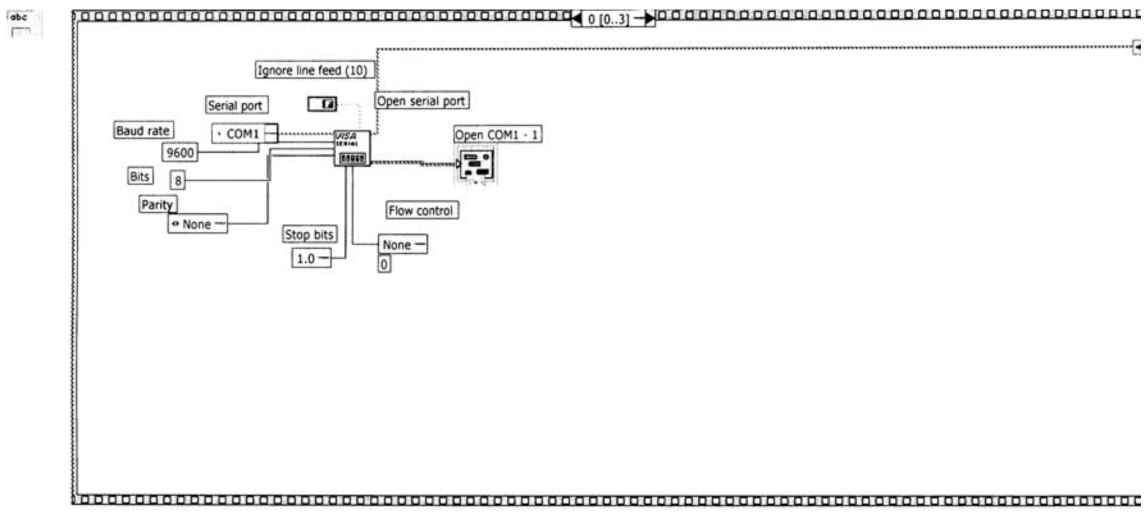

Figure 2 – Shows the part of the program that opens the serial port.

The setup is the following:

Serial port: COM1

Flow control: 0

Stop bits: 1

Parity: None

Bits: 8

Baud rate: 9600

The line feed is set to false so the program does not stop if the ASCII decimal number for line feed (10) is sent.

The long wire to the right allows other parts of the program to use the opened serial port.

4.2 Request profile

The second part of the program can be seen in Figure 3.

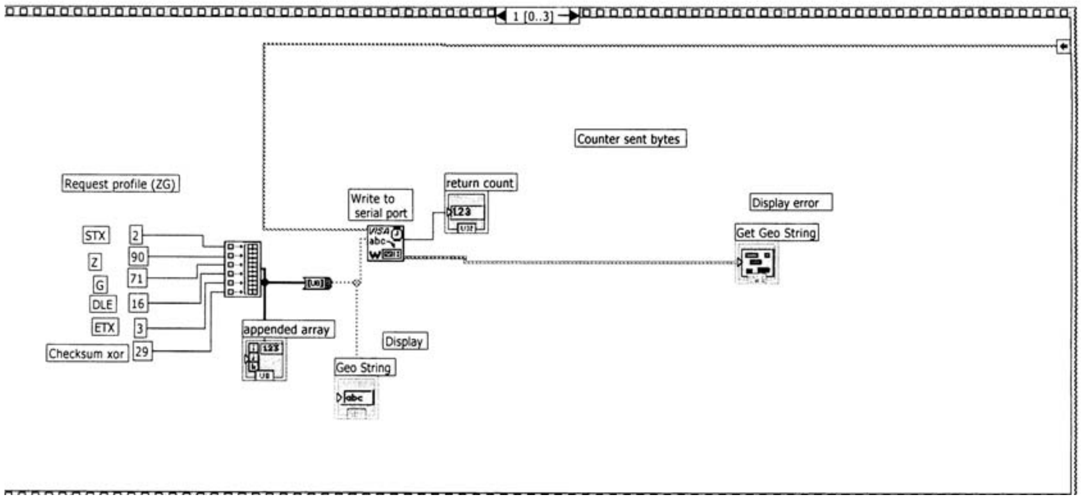

Figure 3 – Shows the part of the program that requests the profile from the sensor.

The Diagram shows how the array to be sent is created and sent. The arrow to the right shows that the program uses a serial port opened in an earlier part of the program.

4.3 Receive profile

Figure 4 shows the part of the program that collects the sent data from the sensor.

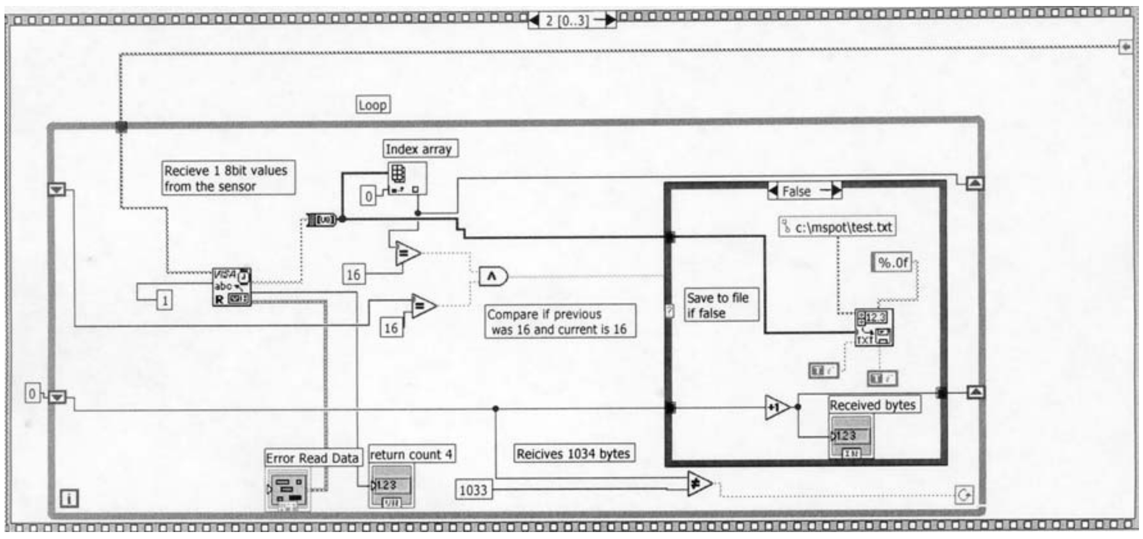

Figure 4 – Shows the part of the program that receives the measurements from the sensor

The left part of the program takes care of the problem if a measured value of 16 (double) is sent. Because of this the quantity of received data will always be the same 1034. The bytes received are stored into the file test.txt in directory c:\mspot.

4.4 Close serial port

When all of the bytes are received the serial port is close to allow other programs to use the port, see Figure 5.

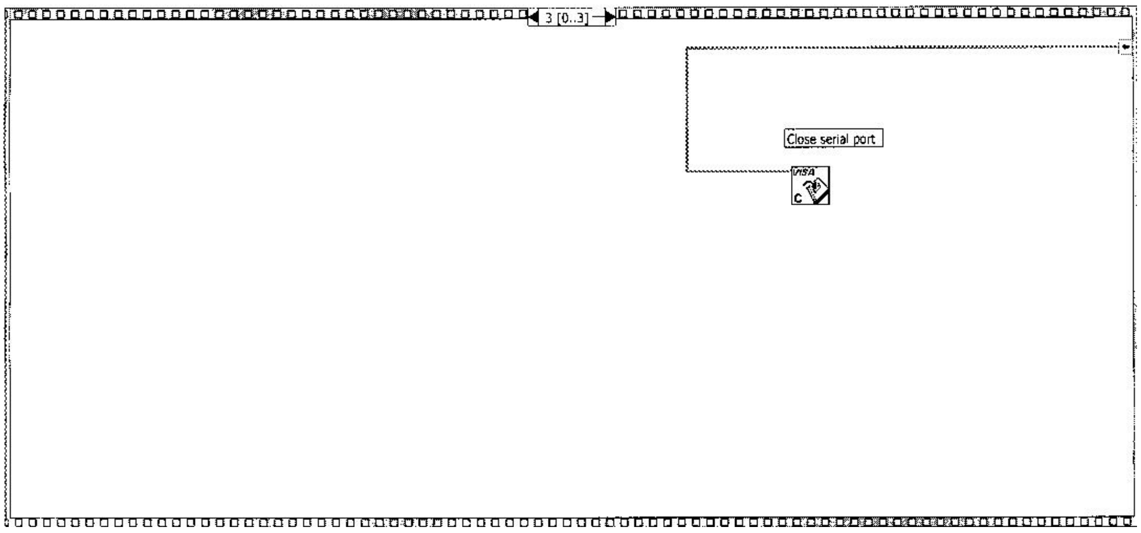

Figure 5 – shows how the used serial port is closed.

4.5 Calculate depth

The program that is used to calculate the depth values from the received measurements can be seen in Figure 6.

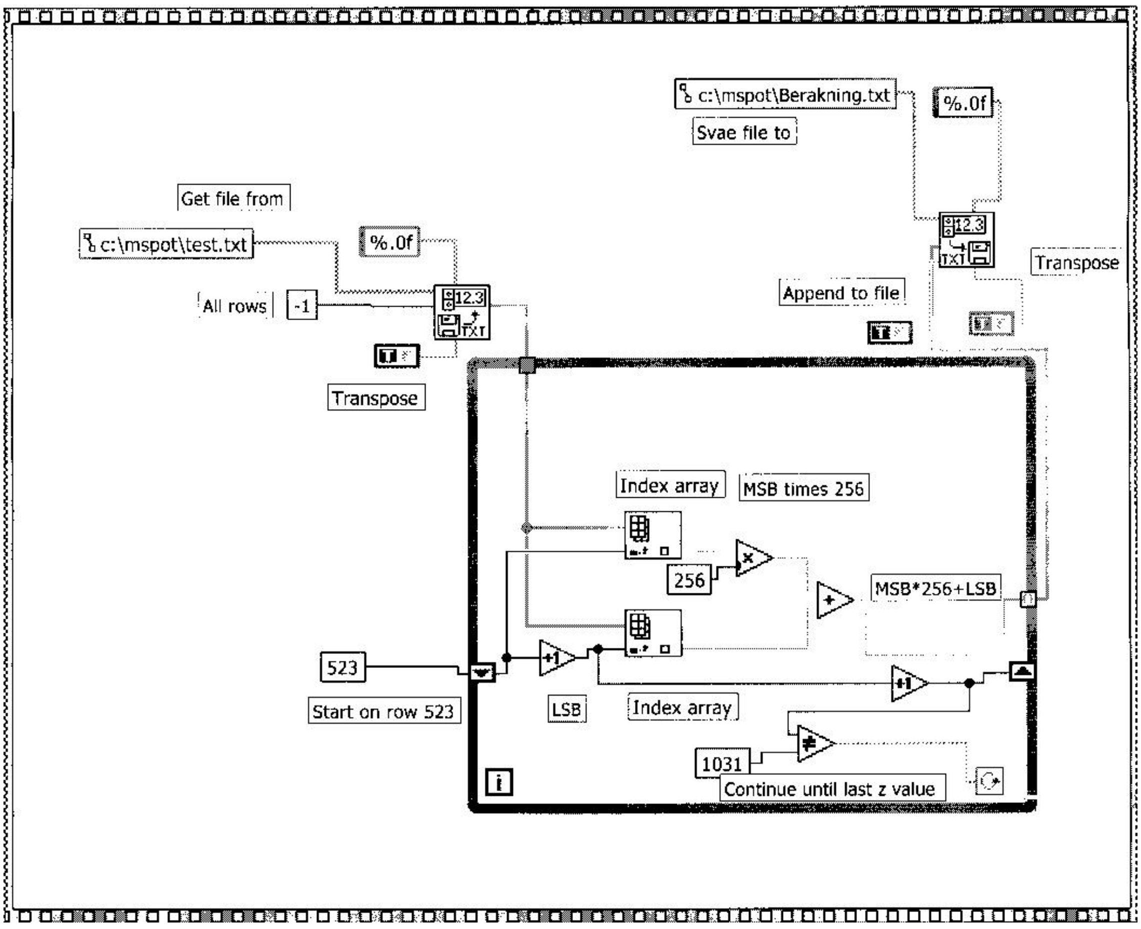

Figure 6 – Contains the program that calculates the depth values from the received bytes

This program reads the values from the saved bytes located in the file test.txt in the c:\mspot directory. The program starts to read at the 523rd line of the file because it contains the first byte describing the height. The 523rd byte is combined with the 524th byte to generate the height of pixel one. The calculated values are stored into the file berakning.txt in the same directory.

5 Result

The result of the work presented in this report is a LABView program that has the ability to send the command ({2}{90}{71}{16}{3}{29}) that makes the sensor transmits the complete geometry measurement. The LABView program also receives the transmission ({2}{122}{status}{t1}{t2}{t3}{t4}{ym0}{yl0}…{ym255}{yl255}

{zm0}{zl0}…{zm255}{zl255}{16}{3}{CKS}) from the sensor and stores the data into a file. Another developed LABView program reads the stored file and calculates the depths of the measurement. The depth-values are stored into a new file.

6 Analysis

The program developed does not know what it is receiving which can cause some troubles, e.g. the sensor is not turned on or the sensor is on but not sensing. This will only cause the program to wait for receiving the predetermined amount of bytes until timeout occurs.
The program does not contain any filter or function to remove noise from the measurement.
The program does not contain any features where high or width is transformed into metric lengths just the values the sensor uses to describe the measurements.

7 Conclusions

The program that has been developed has the ability to command the sensor to send the height measurements, receive the transmission and calculate the height values of the measurement in sensor values.

8 Recommendations for further work

Before the sensor can be used as a geometry sensor for automated robot metal deposition there are still some work left to be done:

- Design a user panel with easy access to the features incorporated in the program
- Program parts for other useful sensor commands like on/off function
- Filter for the measured height
- Edge and height detection
- Determine where the sensor should be used and develop suitable applications
- When must the sensor be returned to Lund

References

M-Spot 90's User manual

Appendix

C-program

```
#include <fcntl.h>
#include <stdio.h>
#include <termios.h>

#define STX  0x02
#define ETX  0x03
#define DLE  0x10

int OpenCSR(char *DeviceName);
int SendCommand(int fd, char *Message, int Len);
int Dump(int fd);
int WaitForProfile(int fd, int *t, int *data);
int WaitForX(int fd, char X);

int main(void)
 {
  int CSR_fd;
  int t[4], data[4096];
  char Message[]={'Z', 'G'};

  if((CSR_fd=OpenCSR("/dev/ttyd1"))==0)
    return(-1);

  printf("Communication port to CSR-4000 open ok\n");

  SendCommand(CSR_fd, Message, sizeof(Message));
  WaitForProfile(CSR_fd, t, data);

  printf("Command sent\n");

  Dump(CSR_fd);

  close(CSR_fd);
  printf("Communication port to CSR-400 closed\n");
  return(0);
 }

int WaitForProfile(int fd, int *t, int *data)
 {
  char ch=0, pch=0;
  int i;

  if(WaitForX(fd, STX)!=1)
    return(0);

  if(WaitForX(fd, 'z')!=1)
    return(0);

  read(fd, &ch, sizeof(ch));
  printf("Staus=%c\n", ch);

  for(i=0; i!=4; i++)
   {
    read(fd, &ch, sizeof(ch));
    t[i]=(int)ch;
    printf("t[%d]=%d\n", i, t[i]);

    if(ch==DLE && pch==DLE)
     {
      i--;
      pch=0;
      continue;
     }

    pch=ch;
   }

  for(i=0; i!=941; i++)
   {
    read(fd, &ch, sizeof(ch));
    data[i]=(int)ch;
    printf("data[%d]=%d\n", i, data[i]);

    if(ch==DLE && pch==DLE)
     {
      i--;
      pch=0;
      continue;
     }

    pch=ch;
   }
/*
  if(WaitForX(fd, DLE)!=1)
    return(0);

  if(WaitForX(fd, ETX)!=1)
    return(0);

  read(fd, &ch, sizeof(ch));
  printf("CKS=%d (0x%X)\n", (int)ch, (int)ch);
*/
  return(1);
 }

int WaitForX(int fd, char X)
 {
  char ch=-1;

  while(ch!=X)
    read(fd, &ch, sizeof(ch));

  return(1);
 }

int Dump(int fd)
 {
  char ch;

  while(1)
   {
    read(fd, &ch, sizeof(ch));
    printf("Received %d %c \n", (int)ch, (char)ch);
   }
 }

int SendCommand(int fd, char *Message, int Len)
 {
  char CKS=0, Start=STX, End[2]={DLE, ETX};
  int i;

  for(i=0; i!=Len; i++)
   CKS^=Message[i];

  printf("CKS=%d (0x%x)\n", (int)CKS, (int)CKS);

  write(fd, &Start, sizeof(Start));
  write(fd, Message, Len);
  write(fd, End, sizeof(End));
  write(fd, &CKS, sizeof(CKS));

  return(0);
 }

int OpenCSR(char *DeviceName)
 {
  int fd;
  struct termios options;

  fd=open(DeviceName, O_RDWR|O_NOCTTY|O_NDELAY);
  if(fd==-1)
   {
    perror("Could not open port to CSR-4000\n");
    return(0);
   }

/* Get current options for the port */
  if(tcgetattr(fd, &options)==-1)
   {
    printf("Can not get current port configuration\n");
    close(fd);
    return(0);
   }

  cfsetispeed(&options, B9600);  /* IN Baudrate */
  cfsetospeed(&options, B9600);  /* OUT Baudrate */
  options.c_cflag |= (CLOCAL|CREAD); /*Enable reciever and set local
mode */
  options.c_cflag &= ~PARENB;  /* 8N1 */
  options.c_cflag &= ~CSTOPB;  /* 8N1 */
  options.c_cflag &= ~CSIZE;   /* 8N1 */
  options.c_cflag |= CS8;      /* 8N1 */
  options.c_cflag &= ~CNEW_RTSCTS;  /* Disable CTS/RTS */
  options.c_iflag &= ~(IXON|IXOFF|IXANY); /* Disable XON /XOFF */
  options.c_lflag &= ~(ICANON|ECHO|ECHOE); /* Canonical input */

  if((tcsetattr(fd, TCSAFLUSH, &options))==-1) /*Flush buffer and set
options */
   {
    printf("Can not set new port configuration\n");
    close(fd);
    return(0);
   }

  fcntl(fd, F_SETFL, 0); /* read blocking behavior */

/*  sleep(2);*/
  tcsetattr(fd, TCSAFLUSH, &options); /*Flush buffer and set options
*/
  return(fd);
 }
```

Working Papers from the University of Trollhättan/Uddevalla

UNIVERSITY
TROLLHÄTTAN • UDDEVALLA

ISSN 1653-2384