Przemysłowy Instytut Automatyki i Pomiarów
Al. Jerozolimskie 202, 02-486 Warszawa

440

DOKUMENT WZORCOWY

ZESPÓŁ INTELIGENTNYCH SYSTEMÓW MOBILNYCH
Nazwa ONB/ZNB

A

Główny wykonawca **Mariusz Kozak**

Wykonawcy:

HEŁM WIRTUALNY JAKO INTERFEJS CZŁOWIEK - MASZYNA STEROWANIE ROBOTEM MOBILNYM

(Tytuł pracy, numer i tytuł etapu)

Zleceniodawca **PIAP**

KIEROWNIK ZESPOŁU
Inteligentnych Systemów Mobilnych
prof. dr hab. inż. Andrzej Masłowski

.....

Pracę zakończono dnia 31.08.98

Nr arch. 7581 **Nr zlecenia** S1858

1

Analiza deskryptorowa

Roboty mobilne, Wirtualna rzeczywistość

Abstrakt

Systemy jakimi są np. roboty mobilne (np. Nomad 200, Khepera, będące obecnie na wyposażeniu ZSM) są wyposażone w kamery video. Sterowanie odbywa się przy pomocy interakcji między operatorem a robotem widzianym bezpośrednio przez operatora. Obecnie coraz częściej zachodzi potrzeba wykorzystania robotów do pracy zdalnej. Wynika to z zastosowania robotów do prac niebezpiecznych oraz szkodliwych dla człowieka. Podczas działania w środowisku o wysokim zagrożeniu konieczna jest precyzyjna kontrola. Kontrolę taką może efektywnie wspomóc odpowiednie zobrazowanie informacji w wirtualnej przestrzeni miejsca pracy robota, stworzonej przy użyciu hełmu wirtualnego, połączone z prezentacją danych pomocniczych.

Tytuły poprzednich sprawozdań

Rozdzielnik

Egz. 1 .....

Egz. 2 .....

Egz. 3 .....

1. Cel pracy

Systemy jakimi są np. roboty mobilne (np. Nomad 200, Khepera, będące obecnie na wyposażeniu ZSM) są wyposażone w kamery video. Sterowanie odbywa się przy pomocy interakcji między operatorem a robotem widzianym bezpośrednio przez operatora. Obecnie coraz częściej zachodzi potrzeba wykorzystania robotów do pracy zdalnej. Wynika to z zastosowania robotów do prac niebezpiecznych oraz szkodliwych dla człowieka. Podczas działania w środowisku o wysokim zagrożeniu konieczna jest precyzyjna kontrola. Kontrolę taką może efektywnie wspomóc odpowiednie zobrazowanie informacji w wirtualnej przestrzeni miejsca pracy robota, stworzonej przy użyciu hełmu wirtualnego, połączone z prezentacją danych pomocniczych.

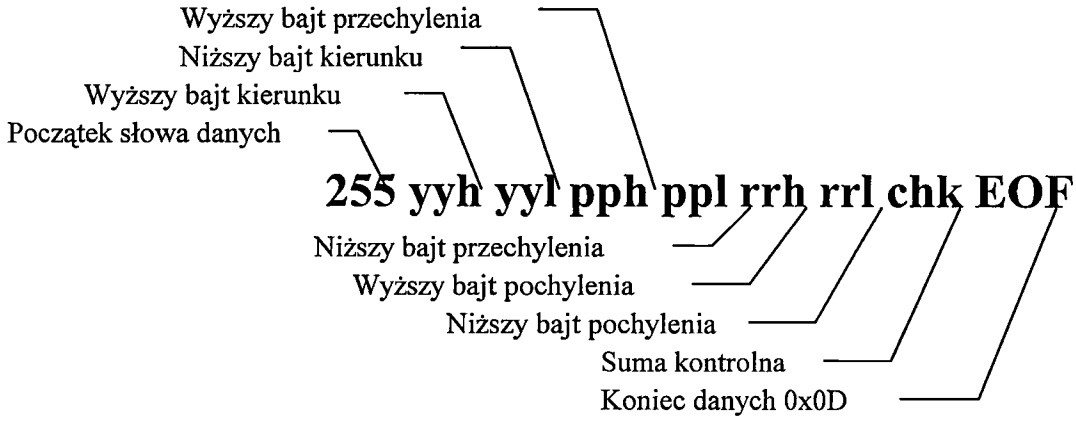
2. Rozwiązanie problemu

Pracę wykonano wykorzystując robota Khepera , hełm wirtualny io-glasses, oraz kartę video TAKREM C210 . Całość oprogramowania została napisana dla systemu Windows95.

a) hełm wirtualny

Jest urządzeniem prezentującym dane z pola pracy robota, oraz urządzeniem śledzącym aktualne położenie głowy operatora. Jest to ustalane na podstawie położenia wektora pola magnetycznego ziemi w układzie związanym z hełmem. Łatwość tego rozwiązania jest okupiona ograniczeniem jakim jest skończoność zakresu pomiarowego (± 180°). Obejście tego problemu jest rozwiązane w komputerze nadrzędnym .
Zastosowany algorytm rozszerza zakres pomiarowy do (±23592960°). Hełm dokonuje pomiaru trzech kątów Eulera. Wykorzystuje się tylko kąt kierunku. Dane o płożeniu hełmu przesyłane są przy pomocy łącz RS232 w formacie tekstowym, po aktywacji trybu pracy, oraz przesłaniu znaku S.

- format danych o położeniu



- słowo trybu pracy hełmu

!M2,P,A,3,3

gdzie:

- ! - początek słowa trybu pracy

- M2 - tryb 8 - bit, kąty Eulera
- P - przesłanie danych po aktywacji (znak „S")
- A - dane tekstowe
- 3,3 - filtracja

Podłączenie wejścia obrazu następuje równolegle z monitorem przy parametrach

- rozdzielczość 640x480
- odświeżanie 50Hz

b.) Robot Khepera

Sterowanie robotem odbywa się w trybie rozkazowym przez drugi port szeregowy.
Komendy są wydawane i przyjmowane w trybie tekstowym.
Wykorzystuje się sterownie wektorem prędkości robota, podając prędkości dla obu kół jezdnych robota. Ograniczoność tych prędkości, oraz konieczność zachowania pozycji (dokładność zliczania kierunku robota) narzuca konieczność stosowania algorytmów nasycenia w komputerze nadrzędnym.
Ponadto z czujników robota pozyskuje się informacje o stanie otoczenia w przestrzeni pracy robota. (8 czujników IRF), i przy pomocy łącza RS232 przesyła do komputera nadrzędnego w postaci liczby 0÷1024.
Formaty komend dla robota:

- prędkość

D,predkość_dla_lewego_sinika, predkość_dla_prawego_sinika,[0x0d]

gdzie:

- predkość_dla_lewego_sinika -1024÷1024
- predkość_dla_prawego_sinika -1024÷1024

- pobranie czujników
  - komenda pobrania

N[0x0d]

  - odpowiedź robota

n,lewy_90,lewy_45,lewy_10,prawy_10,prawy_45,prawy_90,prawy_tylni,lewy_tylni,[0x0d]

c.) Karta video

Pobiera obraz z robota w formacie composit-video, oraz przesyła go do pamięci grafiki po DMA z parametrami:

- rozdzielczość 640x480 (ograniczona rozdzielczością hełmu)
- 16mln kolorów (high - color)

Jest on zamontowana w komputerze nadrzędnym i obsługiwana za pomocą bibliotek SDK dołączonych do karty w postaci DLL.

d.) Komputer nadrzędny

Nadzoruje i synchronizuje pracę całości systemu. Oprogramowanie zainstalowane zostało zaprojektowane wg założeń RAD. Możemy w nim wydzielić cztery podstawowe moduły:

1 - część główna (klasa TForm1)
2 - interfejs hełmu wirtualnego (klasa THelmet)
3 - interfejs robota (Klasa TRobot)
4 - część obrazującą czujniki robota (Klasa TForm2)

ad.1)

- Implementuje obsługę karty video(dołączenie bibliotek, inicjalizacja, zamknięcie)
- Posiada zegar synchronizujący współprace hełmu i robota
  co 100ms:
  - pobiera położenie hełmu
  - wylicza aktualną prędkość hełmu
  - sprawdza ograniczenia na prędkość hełmu
  - pobiera położenie Joysticka
  - wysyła aktualną prędkość obrotową i postępową robota
  - wysyła rozkaz pobrania czujników
  - po przyjęciu zdarzenia pobrania czujników generuje zdarzenie o odświeżeniu wskazań czujników

ad.2)

- posiada własny zegar pobrania położenia hełmu (20ms)
  - wysyła znak pobrania pozycji „S"
  - pobiera i filtruje aktualne położenie hełmu
  - wylicza aktualną prędkość hełmu
- inicjalizuje łącze RS232 (klasa RS232)
- inicjalizuje hełm

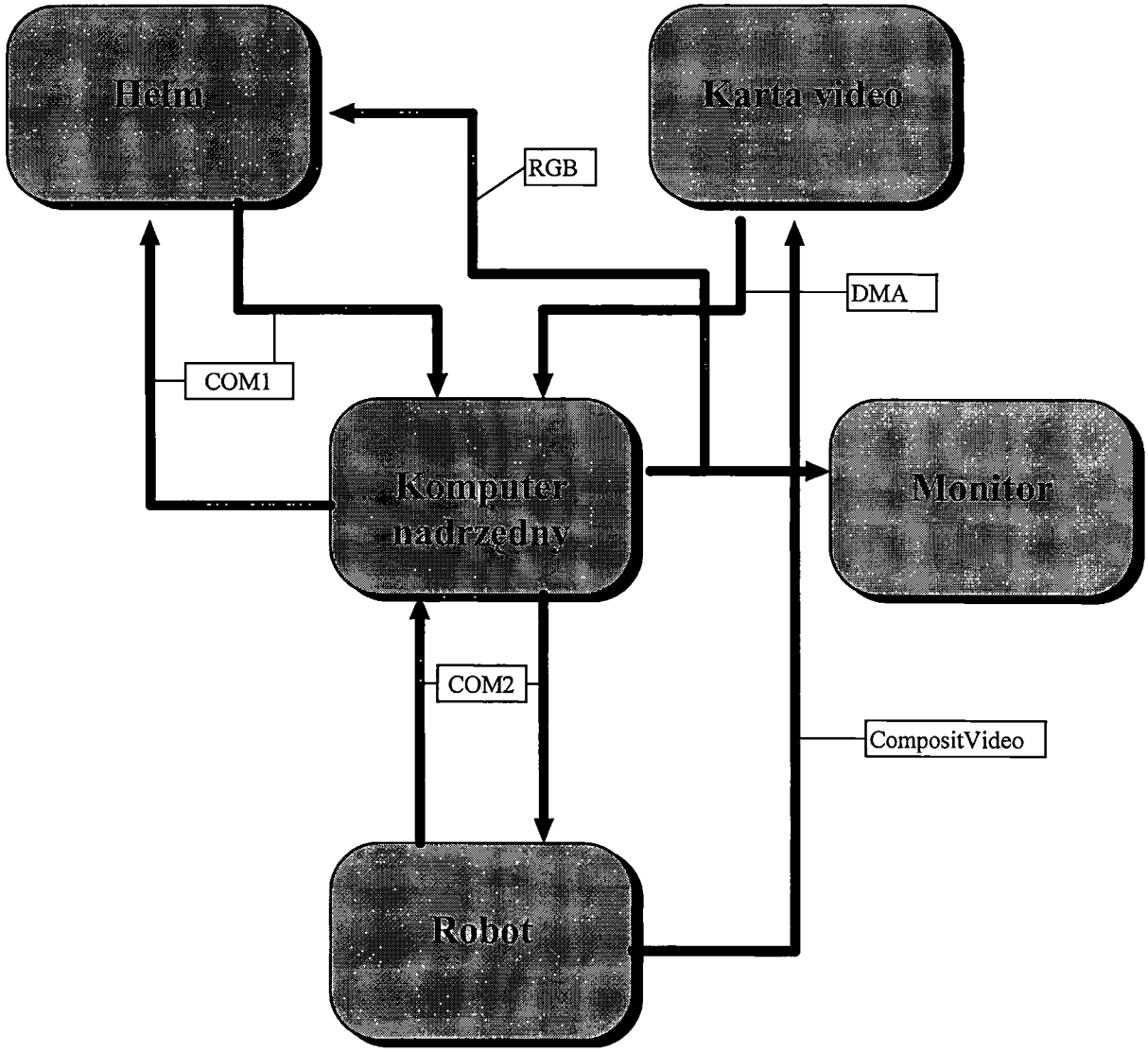
ad.3)

- inicjalizuje złącze RS232 (klasa RS232)
- inicjalizuje robota
- wysyła rozkaz pobrania czujników robota
- wysyła rozkaz dla silników
- pobiera czujniki robota

ad.4)

- wizualizuje czujniki robota(trzy stany odległości od przeszkody)
- przelicza wskazania odległości od przeszkody na skale trójstopniową

Strukturę projektu oraz przepływ danych prezentuje poniższy schemat:



rys1. Schemat blokowy układu i przepływu danych

Pliki źródłowe oprogramowania

1.) dll32.h - moduł obsługi karty video

```
// for proto type declartion
#include <windows.h>
#define ERRCODE     int
#define DLLCALL     pascal

// source type
#define  SRC_TV_TUNER        0
#define  SRC_S_VIDEO         1
#define  SRC_VIDEO           2
#define  SRC_MPEG            3
#define  SRC_VIDEO1          4
#define  SRC_VIDEO2          5
#define  SRC_VIDEO3          6
#define  SRC_VIDEO4          7

// video system
#define  VIDEO_SYSTEM_NTSC     0
#define  VIDEO_SYSTEM_PAL      1
#define  VIDEO_SYSTEM_SECAM    2

// video color
#define  VIDEO_BRIGHTNESS      0
#define  VIDEO_SATURATION      1
#define  VIDEO_HUE             2
#define  VIDEO_SHARPNESS       3
#define  VIDEO_CONTRAST        4

// audio system
#define  AUDIO_VOLUME        0
#define  AUDIO_BASS          1
#define  AUDIO_TREBLE        2
#define  AUDIO_L_CHANNEL     3
#define  AUDIO_R_CHANNEL     4
#define  AUDIO_STEREO        5
#define  AUDIO_MUTE          6

// tv tuner system
#define  MAX_TUNER_SYSTEM      2 // air/cable only
#define  TV_AIR              0
#define  TV_CABLE            1
#define  TV_FAVORITE         2

// cc & teletext
#define  CC_ON               0
#define  CC_OFF              1
#define  TELETEXT_ON         2
#define  TELETEXT_OFF        3

//image capture format
#define  CF_RGB555           0
#define  CF_RGB565           1
```

```
#define CF_RGB888          2
#define CF_RGB888X         3
#define CF_YUY2            4
#define CF_UYVY            5
#define CF_YVU12           6
// used by customer app
typedef struct {
LPSTR lpBmp;      //LPSTR is the buffer to store image data
DWORD dwBufferSize;   //size of the buffer
DWORD dwBytesUsed;   //the actural image size
} DSDATA,FAR* LPDATA;

BOOL TEK_DetectHW();
int DLLCALL (__stdcall *TEK_Init_32)();
void DLLCALL (__stdcall *TEK_DeInit_32)(int );
bool DLLCALL (__stdcall *TEK_SetIniFile_32)(int,char far *);
bool DLLCALL (__stdcall *TEK_GetIniFile_32)(int,char far *);
void DLLCALL (__stdcall *TEK_SaveIniFile_32)(int , char far *);
int DLLCALL (__stdcall *TEK_GetTotalVideoSource_32)(int);
int DLLCALL (__stdcall *TEK_GetIniSourceNum_32)(int);
int DLLCALL (__stdcall *TEK_GetActiveSource_32)(int);
int DLLCALL (__stdcall *TEK_GetSourceInfo_32)(int , int , char far *);
int DLLCALL (__stdcall *TEK_GetSourceSystem_32)(int , int );
void DLLCALL (__stdcall *TEK_SetSourceSystem_32)(int , int,int);
void DLLCALL (__stdcall *TEK_SetActiveSource_32)(int , int );
void DLLCALL (__stdcall *TEK_SetCurInputViewport_32)(int , LPRECT );
void DLLCALL (__stdcall *TEK_GetCurInputViewport_32)(int , LPRECT );
void DLLCALL (__stdcall *TEK_SetVideoColor_32)(int , int , int );
void DLLCALL (__stdcall *TEK_GetVideoColor_32)(int , int ,int far *,int far *, int far * );
bool DLLCALL (__stdcall *TEK_SetOutputWindow_32)(int,HWND,int far * ,int far * ,int far
*,int far *);
void DLLCALL (__stdcall *TEK_GetOutputWindow_32)(int,HWND far *,int far *,int far
*,int far *,int far *);
void DLLCALL (__stdcall *TEK_SetPanAndScroll_32)(int,LPRECT);

BOOL DLLCALL (__stdcall *TEK_Freeze_32)(int);
BOOL DLLCALL (__stdcall *TEK_UnFreeze_32)(int);
BOOL DLLCALL (__stdcall *TEK_FreezeAndGrabFrame_32)(int,int imgFmt);
BOOL DLLCALL (__stdcall *TEK_GrabFrameToBuff_32)(int,char far *,BOOL);
BOOL DLLCALL (__stdcall *TEK_GrabFrameToFile_32)(int,char far *);

//LOW level functions

int DLLCALL TEK_GetAvailableBoardID_32();
BOOL DLLCALL TEK_InitBoard_32(int);
BOOL DLLCALL TEK_CloseBoard_32(int);

void DLLCALL TEK_GetInputViewport_32(int,LPRECT);
void DLLCALL TEK_SetInputViewport_32(int,LPRECT);
void DLLCALL TEK_GetOutputViewport_32(int,int far *,int far *,int far *,int far *);
void DLLCALL TEK_SetOutputViewport_32(int,LPRECT);

BOOL DLLCALL TEK_GetVGAMode_32(int ,int far *);
void DLLCALL TEK_SetCaptureImgType_32(int,int); //0:rgb555, 1:rgb565,2:rgb888
```

```
BOOL DLLCALL TEK_SetFormat_32(int ,int ,int );
//BOOL DLLCALL TEK_GetFormat_32(int ,int ,int far *,int far *);
BOOL DLLCALL TEK_CaptureStart_32(int ,LPDATA );
void DLLCALL TEK_CaptureSingleFrame_32(int ,LPDATA );
BOOL DLLCALL TEK_CaptureFrame_32(int ,LPDATA );
void DLLCALL TEK_CaptureFinish_32(int );

BOOL DLLCALL TEK_SetPreviewMode_32(int,BOOL);
BOOL DLLCALL TEK_SetFormatWindow_32(int,HWND,int far * ,int far * ,int far *,int far
*);

//void DLLCALL TEK_StartGrabWithoutStopping_32(int borad_ID);
//LPBYTE DLLCALL TEK_GetGrabBufPointer_32(int borad_ID);
//void DLLCALL TEK_StopGrab_32(int borad_ID);
```

2.) Moduł obsługi RS232 (klasa RS232)

```
//---------------------------------------------------------------------------
#ifndef rs232H
#define rs232H
//---------------------------------------------------------------------------
#include <vcl\SysUtils.hpp>
#include <vcl\Controls.hpp>
#include <vcl\Classes.hpp>
#include <vcl\Forms.hpp>

//---------------------------------------------------------------------------
typedef void __fastcall (__closure *TReceiveDataEvent)(System::TObject*
Sender,AnsiString& Str);
const PWM_COMMWRITE = WM_USER+1;
class RS232;
//---------------------------------------------------------------------------
class TReadThread : public TThread
{
private:
        TReceiveDataEvent FReceiveDataEvent;
protected:
        void __fastcall Execute();
public:
  HANDLE hCommFile;
        char EvtChar;
        __fastcall TReadThread(bool CreateSuspended);
__published:
        __property TReceiveDataEvent ReceiveDataEvent = { read = FReceiveDataEvent,
write = FReceiveDataEvent};
};
//---------------------------------------------------------------------------
class RS232 : public TComponent
{
private:
  HANDLE    hCommFile;
  WORD              Parity;
```

```
  WORD      ByteSize;
  int       FStopBits;
  int       FBaudRate;
  String    FCommPort;
  DWORD     FMask;
  char      FRxChar;
  TReceiveDataEvent FReceiveDataEvent;
  TReadThread  *ReadThread;
protected:
      void __fastcall CloseReadThread(void);
  void __fastcall GetMessage(TObject *Sender, AnsiString &Str);
public:
      __fastcall RS232(TComponent* Owner);
      bool __fastcall StartComm(void);
      void __fastcall StopComm(void);
  bool __fastcall WriteRS232Data(AnsiString& Str);
      void __fastcall OnMessage(TObject *Sender, AnsiString &Str);
__published:
      __property TReceiveDataEvent ReceiveDataEvent = { read = FReceiveDataEvent,
write = FReceiveDataEvent};
      __property int BaudRate = { read = FBaudRate , write = FBaudRate , default = 9600};
  __property int StopBits = { read = FStopBits , write = FStopBits , default = 0};
  __property String CommPort = { read = FCommPort , write = FCommPort , default =
'COM2'};
  __property DWORD Mask   = { read = FMask , write = FMask , default = EV_RXFLAG |
EV_ERR};
  __property char  RxChar = { read = FRxChar ,write = FRxChar ,default = 'Q'};

};
//---------------------------------------------------------------------------
#endif
```

3.) Moduł obsługi hełmu wirtualnego (klasa THelmet)

- Helmet.h

```
//---------------------------------------------------------------------------
#ifndef helmetH
#define helmetH
//---------------------------------------------------------------------------
#include <vcl\SysUtils.hpp>
#include <vcl\Controls.hpp>
#include <vcl\Classes.hpp>
#include <vcl\Forms.hpp>
#include "rs232.h"
//---------------------------------------------------------------------------
typedef struct TPos {
      signed int Yaw;
  signed int Pitch;
  signed int Roll;
  TPos& operator =(const TPos APosition){
    Yaw   = APosition.Yaw;
    Pitch = APosition.Pitch;
    Roll  = APosition.Roll;
```

```
    }
  TPos& operator -(const TPos APosition){
    Yaw   -= APosition.Yaw;
    Pitch -= APosition.Yaw;
    Roll  -= APosition.Yaw;
    }

} THPosition;
//---------------------------------------------------------------------------
typedef struct H{
        signed long Yaw;
  signed long Pitch;
  signed long Roll;
  H& operator -=(const H A){
    Yaw   -= A.Yaw;
    Pitch -= A.Pitch;
    Roll  -= A.Roll;
    }
  H& operator =(const H A){
    Yaw   = A.Yaw;
    Pitch = A.Pitch;
    Roll  = A.Roll;
    }
  H& operator /=(const int I){
    Yaw   /= I;
    Pitch /= I;
    Roll  /= I;
    }
  H& operator =(const int I)
  {
    Yaw   = I;
    Pitch = I;
    Roll  = I;
    }
  H& operator /(const int I)
  {
    Yaw   /= I;
    Pitch /= I;
    Roll  /= I;
    }
  H& operator +=(const THPosition Y){
    Yaw   += (signed long)(Y.Yaw);
    Pitch += (signed long)(Y.Pitch);
    Roll  += (signed long)(Y.Roll);
    }
  } THelmetPos;
//---------------------------------------------------------------------------
class THelmet : public TComponent
{
private:
        TTimer *Timer1;
  RS232  *RS2321;
  THPosition *MedianList;
  THelmetPos HelmetPos,HelmetPosZero,HelmetSpeed;
```

```
   void __fastcall GetOnePosition(AnsiString &Str);
   void __fastcall SetNoMedian(int ANoMedian);
        void CarryBuffer(void);
   WORD Counter;
   int     FNoMedian;
   int  FGetingInterval;
   String FCommPort;
   int n;
   bool First;
protected:
        void __fastcall OnMessage(TObject *Sender, AnsiString &Str);
   void __fastcall Click(TObject *Sender);
public:
        THelmetPos __fastcall GetPosition(void);
        void __fastcall InitHelmet(void);
   void __fastcall DeInitHelmet(void);
        __fastcall THelmet(TComponent* Owner);
   signed int __fastcall StrToInt(String &Str);
   THelmetPos __fastcall GetSpeed(void);
__published:
        __property int NoMedian = { read = FNoMedian , write = FNoMedian , default = 5};
   __property String CommPort = { read = FCommPort , write = FCommPort , default =
'COM1'};
   __property int GetingInterval = { read = FGetingInterval , write = FGetingInterval , default =
50 };
};
//---------------------------------------------------------------------------
#endif
```

- implementacja klasy THelmet (Helmet.cpp)

```
#pragma link "rs232"
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop

#include "helmet.h"
//---------------------------------------------------------------------------
static inline THelmet *ValidCtrCheck()
{
        return new THelmet(NULL);
}
//---------------------------------------------------------------------------
__fastcall THelmet::THelmet(TComponent* Owner)
        : TComponent(Owner)
{
        n = 0;
   FGetingInterval = 100;
        FNoMedian = 5;
   FCommPort = "COM1";
   RS2321 = new RS232(this);
   Timer1 = new TTimer(this);
   MedianList = new THPosition[NoMedian];
   Timer1->Enabled = false;
```

```
   Timer1->OnTimer = Click;
   Timer1->Interval = GetingInterval;
   RS2321->BaudRate = 9600;
        RS2321->StopBits = 0;
   RS2321->RxChar  = 0x0D;
   First = true;
}
//---------------------------------------------------------------------------
void __fastcall THelmet::InitHelmet(void)
{
RS2321->CommPort = CommPort;
RS2321->StartComm();
RS2321->ReceiveDataEvent = OnMessage;
RS2321->WriteRS232Data("!M2,P,A,3,3\n");
Timer1->Enabled = true;
}
//---------------------------------------------------------------------------
void __fastcall THelmet::Click(TObject *Sender)
{
RS2321->WriteRS232Data("S");
}
//---------------------------------------------------------------------------
void __fastcall THelmet::OnMessage(TObject *Sender, AnsiString &Str)
{
        if (Str[1] == 'O'){}
        else {if (Str[1] == 'E' ) {Timer1->Enabled = false;
                                                                RS2321-
>WriteRS232Data("!M2,P,A,3,3\n");
                        Timer1->Enabled = true;
                                                }
        if(Str[4] == 'F') {GetOnePosition(Str);}
        }
}
//---------------------------------------------------------------------------
void __fastcall THelmet::GetOnePosition(AnsiString &Str)
{
int i,k;

   for(i=1;i<NoMedian;i++){
        MedianList[i-1] = MedianList[i];
     }

   for(k = 1;k<20;k++)
        if((Str[k] == 'F') && (Str[k+1] == 'F')) break;
   MedianList[NoMedian-1].Yaw   = (signed int)(StrToInt(Str.SubString(k+3,4)));
   MedianList[NoMedian-1].Pitch = (signed int)(StrToInt(Str.SubString(k+8,4)));
   MedianList[NoMedian-1].Roll  = (signed int)(StrToInt(Str.SubString(k+13,4)));

        CarryBuffer();

   HelmetSpeed = 0;
   HelmetPos = 0;
        if (Counter < NoMedian) {Counter++;}
        else {for(i=0;i<NoMedian;i++) HelmetPos += MedianList[i];
```

```
                    HelmetPos /= (NoMedian+1);

   for(int i=1;i<NoMedian;i++){
     HelmetSpeed.Yaw += (MedianList[i].Yaw - MedianList[i-1].Yaw);
     HelmetSpeed.Pitch += (MedianList[i].Pitch - MedianList[i-1].Pitch);
     HelmetSpeed.Roll += (MedianList[i].Roll - MedianList[i-1].Roll);
     }
        HelmetSpeed /= (NoMedian-1);
           if (First){
                 HelmetPosZero = HelmetPos;
          First = false;
         }
      }
}
//---------------------------------------------------------------------------
void __fastcall THelmet::SetNoMedian(int ANoMedian)
{
        FNoMedian = ANoMedian;
}
//---------------------------------------------------------------------------
namespace Helmet
{
        void __fastcall Register()
        {
                TComponentClass classes[1] = {__classid(THelmet)};
                RegisterComponents("System", classes, 0);
        }
}
//---------------------------------------------------------------------------
THelmetPos __fastcall THelmet::GetPosition(void)
{
THelmetPos H;
        H = HelmetPos;
   H -= HelmetPosZero;
return(H);
}
//---------------------------------------------------------------------------
void __fastcall THelmet::DeInitHelmet(void)
{
        RS2321->StopComm();
   delete MedianList;
}
//---------------------------------------------------------------------------
int __fastcall THelmet::StrToInt(String &Str)
{
signed long Liczba = 0;

 for(int i = 1;i < Str.Length()+1;i++){
        Liczba *= 16;
        if((Str[i] >=65) && (Str[i] <= 70)) Liczba += Str[i]-55;
        if((Str[i] >=48) && (Str[i] <= 57)) Liczba += Str[i]-48;
   }
  if(Liczba > 32768) Liczba -= 65536;
 return((signed int)(Liczba));
```

```
}
//---------------------------------------------------------------------------
void THelmet::CarryBuffer(void)
{
  if((MedianList[NoMedian-1].Yaw  -  MedianList[NoMedian-2].Yaw)<(-16384-(n*32767)))
n++;
  if((MedianList[NoMedian-1].Yaw - MedianList[NoMedian-2].Yaw)>( 16384-(n*32767))) n-
-;
  MedianList[NoMedian-1].Yaw = MedianList[NoMedian-1].Yaw+(n*32768);
}
//---------------------------------------------------------------------------
THelmetPos __fastcall THelmet::GetSpeed(void)
{
return(HelmetSpeed);
}
```

4.) Moduł obsługi robota (klasa TRobot)

- robot.h

```
//---------------------------------------------------------------------------
#ifndef RobotH
#define RobotH
//---------------------------------------------------------------------------
#include <vcl\SysUtils.hpp>
#include <vcl\Controls.hpp>
#include <vcl\Classes.hpp>
#include <vcl\Forms.hpp>
#include "rs232.h"
//---------------------------------------------------------------------------
typedef int TSensor[8];
//---------------------------------------------------------------------------
typedef void __fastcall (__closure *TReceiveSensor)(System::TObject* Sender,TSensor&
Sensor);
//---------------------------------------------------------------------------
class TRobot : public TComponent
{
private:
        TSensor Sensor;
        RS232 *RS2321;
  String FCommPort;
  DWORD  FBaudRate;
  TReceiveSensor FReceiveSensor;
        void __fastcall ReadSensor(String& Str);
protected:
        void __fastcall GetMessage(TObject *Sender, AnsiString &Str);
public:
        __fastcall TRobot(TComponent* Owner);
        void __fastcall InitRobot(void);
        void __fastcall DeInitRobot(void);
        void __fastcall SetSpeed(int TranslationSpeed, int RotationSpeed);
        void __fastcall GetSensor(void);
```

```
        void __fastcall Stop(void);
__published:
        __property String CommPort = { read = FCommPort , write = FCommPort , default =
'COM2'};
   __property DWORD BaudRate = {read = FBaudRate , write = FBaudRate , default =
38400};
   __property TReceiveSensor ReceiveSensor = { read = FReceiveSensor , write =
FReceiveSensor };
};
//---------------------------------------------------------------------------
#endif
```

- implementacja modułu TRobot (robot.cpp)

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop
#pragma link "rs232"

#include "Robot.h"
//---------------------------------------------------------------------------
static inline TRobot *ValidCtrCheck()
{
        return new TRobot(NULL);
}
//---------------------------------------------------------------------------
__fastcall TRobot::TRobot(TComponent* Owner)
        : TComponent(Owner)
{
        FCommPort = "COM2";
   FBaudRate = 38400;
        RS2321 = new RS232(this);
}
//---------------------------------------------------------------------------
namespace Robot
{
        void __fastcall Register()
        {
                TComponentClass classes[1] = {__classid(TRobot)};
                RegisterComponents("System", classes, 0);
        }
}
//---------------------------------------------------------------------------
void __fastcall TRobot::DeInitRobot(void)
{
        Stop();
        RS2321->StopComm();
}
//---------------------------------------------------------------------------
void __fastcall TRobot::InitRobot(void)
{
        RS2321->BaudRate = BaudRate;
```

```
        RS2321->CommPort = CommPort;
        RS2321->RxChar   = 0x0d;
        RS2321->StartComm();
        RS2321->ReceiveDataEvent = GetMessage;
}
//---------------------------------------------------------------------------
void __fastcall TRobot::Stop(void)
{
        RS2321->WriteRS232Data("D,0,0\n");
}
//---------------------------------------------------------------------------
void __fastcall TRobot::SetSpeed(int TranslationSpeed, int RotationSpeed)
{
int motor_left,motor_right;

        motor_left  = TranslationSpeed - RotationSpeed;
        motor_right = TranslationSpeed + RotationSpeed;
        RS2321-
>WriteRS232Data("D,"+IntToStr(motor_left)+","+IntToStr(motor_right)+"\n");
}
//---------------------------------------------------------------------------
void __fastcall TRobot::GetSensor(void)
{
        RS2321->WriteRS232Data("N\n");
}
//---------------------------------------------------------------------------
void __fastcall TRobot::GetMessage(TObject *Sender, AnsiString &Str)
{
        for(int i = 0;i<Str.Length();i++){
    switch(Str[i]){
        case 'n' :
        ReadSensor(Str);
       return;
                case 'd' :
        return;
     default :
         break;
     }
  }
}
//---------------------------------------------------------------------------
void __fastcall TRobot::ReadSensor(String& Str)
{
char Message[256];
char tokensep[] = ",";

        strcpy(Message,Str.c_str());
  strtok(Message,tokensep);

  for(int i = 0;i<8;i++){
        Sensor[i] = atoi(strtok(NULL,tokensep));}

        if (ReceiveSensor) ReceiveSensor(this,Sensor);
}
```

```
//---------------------------------------------------------------------------
```

5.) Moduł wizualizacji czujników (klasa TForm2)

- sensory.h

```
//---------------------------------------------------------------------------
#ifndef sensoryH
#define sensoryH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include "okno.h"
#include <vcl\ExtCtrls.hpp>
//---------------------------------------------------------------------------
class TForm2 : public TForm
{
__published:    // IDE-managed Components
        TShape *Shape1;
        TShape *Shape2;
        TShape *Shape3;
        TShape *Shape4;
        TShape *Shape5;
        TShape *Shape6;
        TShape *Shape7;
        TShape *Shape8;
        TShape *Shape9;
        TShape *Shape10;
        TShape *Shape11;
        TLabel *Label1;
private: // User declarations
        void __fastcall GetSensor(TObject *Sender, TSensor &Sensor);
   TColor Kolor[8];
public:         // User declarations
        __fastcall TForm2(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern TForm2 *Form2;
//---------------------------------------------------------------------------
#endif
```

- implementacja TForm2 (sensory.cpp)

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop

#include "sensory.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
```

```
TForm2 *Form2;
//---------------------------------------------------------------------------
__fastcall TForm2::TForm2(TComponent* Owner)
        : TForm(Owner)
{
        Form1->Robot1->ReceiveSensor = GetSensor;
}
//---------------------------------------------------------------------------
void __fastcall TForm2::GetSensor(TObject *Sender, TSensor &Sensor)
{
for(int i = 0;i<8;i++){
        if(Sensor[i] < 10){Kolor[i] = clGreen;}
        else{
        if(Sensor[i] <1000){Kolor[i] = clYellow;}
        else{Kolor[i] = clRed;}
        }
  }

Shape2->Brush->Color = Kolor[2];
Shape3->Brush->Color = Kolor[1];
Shape4->Brush->Color = Kolor[0];
Shape5->Brush->Color = Kolor[7];
Shape6->Brush->Color = Kolor[6];
Shape7->Brush->Color = Kolor[5];
Shape8->Brush->Color = Kolor[4];
Shape9->Brush->Color = Kolor[3];
}
//---------------------------------------------------------------------------
```

6.) Moduł źródłowy programu głównego (klasa TForm1)

```
//---------------------------------------------------------------------------
#ifndef proba_newH
#define proba_newH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include <mmsystem.h>
#include "robot.h"
#include "rs232.h"
#include "Helmet.h"
#include "dll32.h"
//#include "sensory.h"
#include <vcl\ExtCtrls.hpp>

#define MAX_ROTATE  20
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published:    // IDE-managed Components
        TRobot *Robot1;
        THelmet *Helmet1;
```

```
        TTimer *Timer1;
        void __fastcall FormCreate(TObject *Sender);
        void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
        void __fastcall Timer1Timer(TObject *Sender);


private: // User declarations
        JOYINFO Joy_XY_1;
        int Position[2];
   int Translation_speed,Rotation_speed;
        void __fastcall GetSensor(TObject *Sender, TSensor &Sensor);
   HANDLE dll;
   int no;
public:         // User declarations
        __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern TForm1 *Form1;
//extern TForm2 *Form2;
//---------------------------------------------------------------------------
#endif
```

- implementacja programu głównego (klasa TFomr1)

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop

#include "proba_new.h"
//#include "sensory.h"
//---------------------------------------------------------------------------
#pragma link "robot"
#pragma link "rs232"
#pragma link "Helmet"
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Robot1->ReceiveSensor = GetSensor;
for(int i = 0;i<2;i++) Position[i] = 0;
dll = NULL;
dll = LoadLibrary("tekdci32.dll");
  if (dll != NULL) {
        TEK_DeInit_32       =       (void       DLLCALL       (__stdcall       *)(int
))GetProcAddress(dll,"TEK_DeInit_32");
        TEK_Init_32 = (int  DLLCALL (__stdcall *)())GetProcAddress(dll,"TEK_Init_32");
```

```
        TEK_SetOutputWindow_32 = (bool DLLCALL (__stdcall *)(int,HWND,int far * ,int
far * ,int far *,int far *))GetProcAddress(dll,"TEK_SetOutputWindow_32");
   }
 if (TEK_Init_32){
                no = TEK_Init_32();

TEK_SetOutputWindow_32(no,Handle,&ClientRect.Left,&ClientRect.Top,&Width,&Height);
                }
Helmet1->InitHelmet();
Robot1->InitRobot();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
TEK_DeInit_32(no);
Helmet1->DeInitHelmet();
Robot1->DeInitRobot();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::GetSensor(TObject *Sender, TSensor &Sensor)
{
//++
//Form2->Show();
//Form2->Label1->Caption = IntToStr(Sensor[0]);
//Form2->Label2->Caption = IntToStr(Sensor[1]);
//++
if(joyGetPos(JOYSTICKID1,&Joy_XY_1)==JOYERR_NOERROR){
        Translation_speed = Joy_XY_1.wYpos-34728;
        Robot1->SetSpeed(-Translation_speed/2000,Rotation_speed);}
else{Close();}
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
        Robot1->GetSensor();
        Position[0] = Position[1];
//      Position[1] = Helmet1->GetPosition().Yaw/100;
        Rotation_speed = Position[1] - Position[0];
if(Rotation_speed > MAX_ROTATE){
        Rotation_speed = MAX_ROTATE;
        Position[1] = Position[0] + MAX_ROTATE;
   }
if(Rotation_speed < (-MAX_ROTATE)){
        Rotation_speed = (-MAX_ROTATE);
        Position[1] = Position[0] - MAX_ROTATE;
   }
}
//---------------------------------------------------------------------------
```