A

Zakupy sprzętu komputerowego w tematach wykonywanych we współpracy z EWG - projekty: IMPACT-PECO zlec. 1482E INGRID-PECO zlec. 1484E

1. Komputer do tematu IMPACT:

- Płyta główna - Pentium 90 MHz, 16 MB RAM
- SVGA 2MB S3
- Monitor kolor 15"
- Filtr polaroid
- FDD 1.44/1.2 MB
- 2xHDD 350 MB
- 4xRS232 + 1xParallel
- Mysz + tablet
- Złącze przejściowe 9/25
- Karta Ethernet

2. Komputer do tematu INGRID:

- Notebook, płyta główna 486DX/33, 4 MB RAM
- FDD 1.44 MB
- HDD 200MB
- 2xRS232 + 1xParallel
- Mysz + tablet
- Karta grafiki SVGA kolor 512 kB
- Monitor LCD 10"
- Dodatkowy monitor kolor 15"
- Dodatkowa klawiatura
- Dodatkowa obudowa typu PC

3. Modernizacja komputera 286/16/8:

- Sterownik FDD/HDD
- FDD 1.4 MB
- HDD 250 MB

4. Komputer 286/12:

- Płyta główna 386DX/40, 4 MB RAM
- karta grafiki SVGA 512 kB
- Monitor kolor 14"
- Sterownik FDD/HDD
- FDD 1.44"
- HDD 250 MB
- Mysz + tablet

5. Komputer 286/10:

- Płyta główna 486DX2/50, 8 MB RAM
- karta grafiki SVGA 512 kB
- Monitor kolor 15"

- Sterownik FDD/HDD
- FDD 1.44"
- HDD 250 MB
- Klawiatura
- Mysz + tablet

6. Komputer 386:

- Sterownik FDD/HDD
- HDD 250 MB
- Wymiana płyty głównej 386DX/40, 4 MB RAM

7. Komputer 486:

- HDD 250 MB

Oprogramowanie:

- do poz. 1., 2., 4., 5.: DOS + Windows
- do poz. 1., 2.: Word for Windows + Norton Commander 4.0

ZAŁĄCZNIK 5.

do sprawozdania nr rej. 7145

Oprogramowanie rozwiązujące proste i odwrotne zadanie kinematyczne dla robota IRb-6 pięcioosiowego - listing.

Lista procedur:

- for_back.c - program główny - rozwiązuje proste, a następnie odwrotne zadanie kinematyczne i drukuje wyniki dla porównania. Danymi wejściowymi są współrzędne wewnętrzne podane w stopniach,
- dtoi - zamienia liczby z formatu double na integer,
- extconst - stałe wykorzystywane w programie,
- imgmat - pomocnicze procedury matematyczne,
- intorad.c - przelicza inkrementy na radiany
- it_back1.c - rozwiązuje odwrotne zadanie kinematyczne; wejście - macierz przejścia od układu podstawy do układu kołnierza, wyjście - tablica sinusów i casinusów kątów w poszczególnych osiach.
- m_irp6_2 - rozwiązuje proste zadanie kinematyczne,
- naglowek - drukuje nagłówek, datę i czas symulacji.,
- rad_to_in - przelicza radiany na inkrementy,

```
/* Sprawdzenie rozwiazywania prostego i odwrotnego zadania kinematycznego    *
 * dla robota IRp-6 z porownaniem wynikow                                     */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <mem.h>
#include "index.h"

extern void ptell(char bufor1[122], int *liczba, char *wart[20]);

static double dai[6], ai[6], t_mac[4][4];
static double c[6], s[6];
static int aac;
static char buf[122], *aav[20];
double ss[6], cc[6];

main ( int ac, char *av[] )
{
  FILE *fpi;
  extern double L[];
  extern double eps;
  long inkr2[6];
  int i;

      if ( ac < 2 ) {
              fprintf ( stderr, "Usage: inout infile [ outfile ]\n" );
      }
      if (( fpi = fopen ( av[1], "rt" )) == NULL ) {
              perror ( "infile fopen" );
              return ( 1 );
      }
      if ( ac >= 3 ) {
              if ( freopen ( av[2], "wt", stdout ) == NULL ) {
                      perror ( "freopen " );
                      return ( 1 );
              }
      }

  for ( ;; ) {
              fgets ( buf, 120, fpi );
              if ( ferror ( fpi )) {
                      perror ( "fgets" );
                      return ( 1 );
              }
              if ( feof ( fpi ))
                      break;
              if ( buf[0] == '*' || buf[0] == '#' )
                      continue;
              ptell ( buf, &aac, aav );
              if ( aac == 0 )
                      continue;
              memset ( dai, '\0', sizeof ( dai ));

  pisz_naglowek ();
  printf ("***           Proste i odwrotne zadanie ");
  printf ("kinematyczne IRp-6                    ***\n");
  printf ("*-------------------------------------");
  printf ("-------------------------------------*\n");
  printf ("Parametry wejsciowe - wspolrzedne wewnetrzne robota\n");
  printf ("+-------------------------------------");
```

```
printf ("---------------------------------------+\n");
printf ("| (we) TETA[rad] : |");
for ( i = 0; i < aac && i < 5; i++ ) {
  dai[i] = atof ( aav[i] );
  ai[i] = deg_to_rd ( dai[i] );
  printf (" %8.6lf  |",ai[i]);
}
printf ("\n");
printf ("|---------------------------------------");
printf ("---------------------------------------|\n");
printf ("|      TETA[deg] : |");
for ( i = 0; i < 5; i++ )
  printf (" %8.6lf  |",dai[i]);
printf ("\n");
printf ("|---------------------------------------");
printf ("---------------------------------------|\n");

for ( i = 0; i < 5; i++ ) {
  c[i] = cos ( ai[i] );
  s[i] = sin ( ai[i] );
}

rad_to_in_irp_6 (inkr2, c, s);
printf ("|    poswew[ink] : |");
for (i = 0; i < 5; i++)
  printf (" %8ld  |",inkr2[i]);
printf ("\n");
printf ("+---------------------------------------");
printf ("---------------------------------------+\n");

make_irp6_mac2 (c, s, t_mac);

printf (" Rozwiazanie prostego zadanie kinematycznego - macierz pozycji: \n");
for (i = 0; i < 4; i++)
  printf ("| %8.2lf | %8.2lf | %8.2lf | %8.2lf\n", t_mac[i][0], t_mac[i][1],
          t_mac[i][2], t_mac[i][3]);
printf ("+---------------------------------------");
printf ("---------------------------------------+\n");

printf (" Rozwiazanie odwrotnego zadanie kinematycznego: \n");

it_back_6 (t_mac, ss, cc, ai);
printf ("| (wy) TETA[rad] : |");
for ( i = 0; i < 5; i++ )
  printf (" %8.6lf  |",ai[i]);
printf ("\n");
printf ("|---------------------------------------");
printf ("---------------------------------------|\n");
printf ("|      TETA[deg] : |");
for ( i = 0; i < 5; i++ ) {
  dai[i] = rd_to_deg(ai[i]);
  printf (" %8.6lf  |",dai[i]);
}
printf ("\n");
printf ("|---------------------------------------");
printf ("---------------------------------------|\n");

rad_to_in_irp_6 (inkr2, cc, ss);
printf ("|    poswew[ink] : |");
for (i = 0; i < 5; i++)
  printf (" %8ld  |",inkr2[i]);
```

```
  printf ("\n");
  printf ("+--------------------------------------");
  printf ("--------------------------------------+\n");
  printf ("\n\n\n\n\n");
  } /* for ( ; ; ) */

    return ( 0 );
}
```

```
/*
 *  dtoi.c            SKG  30-Oct-1988
 *  Skarzysko              May-28-1990
 *
 *    convert double to int with rounding
 */

#include <math.h>

int dtoi ( x )
double x;
{
    if ( x > 0. )
   return ( (int)( x + .5 ));
    else
   return ( (int)( x - .5 ));
}

long dtol ( x )
double x;
{
   if ( x > 0. )
   return ( (long)( x + .5 ));
    else
   return ( (long)( x - .5 ));
}
```

```
/* stale wykorzystywane przy przeliczaniu modelu kinematyki robota IRp-6 */
double
/* wspolczynnik przelozenia silnik-ramie w osi pierwszej */
   wsp1 = 1.55340e-04,
/* wspolczynnik przelozenia silnik-ramie w osi czwartej */
   wsp4 = 1.91748e-04,
/* wspolczynnik przelozenia silnik-ramie w osi piatej */
   wsp5 = 3.22943e-04,
/* kat pomiedzy gornym ramieniem pantografu a jego stala przekatna dla osi *
 * drugiej w jej zerowym polozeniu                                         */
   beta_2 = 1.38375e+00,
/* kat pomiedzy gornym ramieniem pantografu a jego stala przekatna dla osi *
 * trzeciej w jej zerowym polozeniu                                        */
   beta_3 = 8.60148e-01,
/* ramiona pantografu - czwarty bok stanowi sruba przekladni  */
   L1 = 140.,
   L2 = 157.5,
   L3 = 183.,
/* dlugosc sruby przekladni osi drugiej w polozeniu zerowym */
   L02 = 2.55585e+02,
/* dlugosc sruby przekladni osi trzeciej w polozeniu zerowym */
   L03 = 1.83835e+02,
/* przekatna pantografu - druga przekatna jest zmienna i zalezy od dlugosci *
 * sruby                                                                    */
   L23 = 2.41444e+02,
   LL2 = 7.78952e+04,   /* L1 * L1 + L23 * L23 */
   _2LL = 6.76043e+04;  /* 2 * L1 * L23 */
/* przesuw sruby przekladni w osi 2 i 3 przy obrocie silnika o 1 inkrement */
/* c5_256 = (skok sruby - 5mm) / (jeden obrot walu silnika - 256inkr.)     */
double c5_256 = 0.01953125000000000;

double L[5] = { 0., 450., 670., 0., 0.};

double eps = 0.001;

long synchrpos[5] = {19112L, 4580L, 2808L, 1187L, 2344L};
long default_geom_pos[5] = {19112L, 4580L, 2808L, 8196L, 9191L};
```

```
/*************************************************************************
 * imgmat.c *       * SKG *     * 22-Jun-1987 *       * poprawki 25.02.1992. *
 *************************************************************************
 * useful matrix and vector operations                                   *
 *-----------------------------------------------------------------------*
 * procedury operacji na macierzach i inne funkcje przydatne w rozwiazywaniu *
 * zadan kinematyki dla robota IRp-6                                     *
 *************************************************************************/

#include <stdio.h>
#include <math.h>
#include "imgmat.h"

void mat_prod ( x, y, r, m, p, n )
```

```
double *x, *y;              /* input matrices              */
double *r;              /* for resultant matrix       */
int m;                  /* no of rows of matrix x          */
int p;                  /* no of columns of x and rows of y     */
int n;                  /* no of columns of y              */
{
    int i, j, k;
    double *d;

    for ( i = 0; i < m; i++ ) {
        for ( j = 0; j < n; j++ ) {
            d = r + i * n + j;
            *d = 0.;
            for ( k = 0; k < p; k++ ) {
                *d += *( x + i * p + k ) * *( y + k * n + j );
            }
        }
    }
}

void mat_tra ( a, b, m, n )
double *a, *b;
int m, n;
{
    int i, j;

    for ( i = 0; i < m; i++ )
        for ( j = 0; j < n; j++ )
            *( b + j * m + i ) = *( a + i * n + j );
}

double norm ( x, n )
double *x;
int n;
{
    int i;
    double r = 0;

    for ( i = 0; i < n; i++ )
        r += x[i] * x[i];

    return ( sqrt ( r ));
}

void norm_vec ( pd, n )
double *pd;
int n;
{
    int i;
    double an;

    if (( an = norm ( pd, n )) < .000001 )
         return;
    for ( i = 0; i < n; i++, pd++ )
        *pd /= an;
}

double scalar_prod ( px, py, n )
double *px, *py;
int n;
{
```

```
    double sum = 0.;
    int i;

    for ( i = 0; i < n; i++, px++, py++ )
        sum += *px * *py;
    return ( sum );
}

void vector_prod ( px, py, pz, n )
double *px, *py, *pz;
int n;
{
    switch ( n ) {
         case 1    :    *pz = *px * *py;
                        break;
         case 2    :    *pz = px[0] * py[1] - px[1] * py[0];
                        break;
         case 3    :    pz[0] = px[1] * py[2] - px[2] * py[1];
                        pz[1] = px[2] * py[0] - px[0] * py[2];
                        pz[2] = px[0] * py[1] - px[1] * py[0];
                        break;
         default   :    printf ( "As for now no vetcor product for dim > 3\n" );
                        break;
    }
}

void add_vec ( double *x, double *y, double *z, int n )
{
    int i;

    for ( i = 0; i < n; i++ )
        z[i] = x[i] + y[i];
}

void sub_vec ( double *x, double *y, double *z, int n )
{
    int i;

    for ( i = 0; i < n; i++ )
        z[i] = x[i] - y[i];
}

void scalar_by_vec ( double s, double *x, double *y, int n )
{
    int i;

    for ( i = 0; i < n; i++ )
        y[i] = s * x[i];
}

void add_matr ( double *a, double *b, double *c, int m, int n )
{
    int i, j, k;

    for ( i = 0; i < m; i++ )
        for ( j = 0; j < n; j++ ) {
            k = i * n + j;
            c[k] = a[k] + b[k];
        }
}
```

```
void sub_matr ( double *a, double *b, double *c, int m, int n )
{
    int i, j, k;

    for ( i = 0; i < m; i++ )
        for ( j = 0; j < n; j++ ) {
            k = i * n + j;
            c[k] = a[k] - b[k];
        }
}

double matr_n_inf_norm ( double *a, int n )
{
    double d, max;
    int i, j, k;

    max = fabs ( a[0] );
    for ( i = 0; i < n; i++ )
        for ( j = 0; j < n; j++ ) {
            k = i * n + j;
            d = fabs ( a[k] );
            if ( d > max )
                max = d;
        }
    return max;
}

/* Arcus tangens z rozszerzeniem obszaru wokol zera */
double latan2 ( double y, double x )
{
    if ( fabs ( x ) < EPSILON && fabs ( y ) < EPSILON )
        return ( 0. );

    return ( atan2 ( y, x ));
}
```

```
#include <stdio.h>
#include <math.h>
#include "index.h"

int in_to_rad_irp_6 ( poswew, c, s )
long poswew[];
double c[], s[];
{
    double x, y, z;
    double L2e, L3e;

    x = wsp1 * (double)(poswew[_1] - default_geom_pos[_1]);
    s[_1] = sin ( x );
    c[_1] = cos ( x );

    x = wsp4 * (double)(poswew[_4] - default_geom_pos[_4]);
    s[_4] = sin ( x );
    c[_4] = cos ( x );

    x = wsp5 * (double)(poswew[_4] - default_geom_pos[_4]
        - poswew[_5] + default_geom_pos[_5]);
    s[_5] = sin ( x );
    c[_5] = cos ( x );

    L2e = L02 + (poswew[_2] - default_geom_pos[_2]) * c5_256;
    y = (LL2 - (L2e * L2e)) / _2LL;
    z = acos (y);
    x = beta_2 - z;
    s[_2] = sin ( x );
    c[_2] = cos ( x );

    L3e = L03 + (poswew[_3] - default_geom_pos[_3]) * c5_256;
    y = (LL2 - (L3e * L3e)) / _2LL;
    z = acos (y);
    x = beta_3 - z;
    s[_3] = sin ( x );
    c[_3] = cos ( x );

    return ( 0 );
}
```

```
/*****************************************************************************
 *                            it_back_6                                      *
 *****************************************************************************
 *                                                                           *
 * Obliczanie sinusow i cosinusow katow w poszczegolnych osiach robota IRp-6 *
 * na podstawie macierzy pozycji ukladu wspolrzednych zwiazanego z kolnierzem*
 *                                                                           *
 *      it - wejsciowa macierz pozycji kartezjanskiej T_MAC                  *
 *      s  - tablica sinusow katow osi robota                                *
 *      c  - tablica cosinusow katow osi robota                              *
 *****************************************************************************/

#include <stdio.h>
#include <math.h>
#include "index.h"

int it_back_6 (it, s, c, teta)
double it[][4], s[], c[], teta[];
{
double beta, fi, gamma, w, ss;

  teta[_1] = latan2(it[_2][_4], it[_1][_4]);
  c[_1] = cos(teta[_1]);
  s[_1] = sin(teta[_1]);

  ss = sqrt(it[_1][_4] * it[_1][_4] + it[_2][_4] * it[_2][_4]);
  fi = latan2(it[_3][_4], ss);
  w = sqrt(it[_3][_4] * it[_3][_4] + ss * ss);
  beta = acos((L[_2] * L[_2] + w * w - L[_3] * L[_3]) / (2. * L[_2] * w));
  teta[_2] = -M_PI_2 + beta + fi;
  c[_2] = cos(teta[_2]);
  s[_2] = sin(teta[_2]);

  gamma = acos((L[_2] * L[_2] + L[_3] * L[_3] - w * w) / (2. * L[_2] * L[_3]));
  teta[_3] = -M_PI_2 + gamma + teta[_2];
  c[_3] = cos(teta[_3]);
  s[_3] = sin(teta[_3]);

  teta[_4] = latan2(it[_3][_3], (it[_1][_3] * c[_1] + it[_2][_3] * s[_1]));
  c[_4] = cos(teta[_4]);
  s[_4] = sin(teta[_4]);

  teta[_5] = latan2((it[_1][_1] * s[_1] - it[_2][_1] * c[_1]),
                    (it[_1][_2] * s[_1] - it[_2][_2] * c[_1]));
  c[_5] = cos(teta[_5]);
  s[_5] = sin(teta[_5]);

        return (0);
}
```

```
#include <time.h>
#include <stdio.h>

void pisz_naglowek ()
{
    struct tm *czas_akt;
    time_t czas_sek;

/* Naglowek do wydruku wynikowego */

    time (&czas_sek);
    czas_akt = localtime (&czas_sek);

    printf ("****************************************");
    printf ("****************************************\n");
    printf ("*****                 SYMULACJA KINEMATKI");
    printf (" ROBOTA IRp-6                        *****\n");
    printf ("****************************************");
    printf ("****************************************\n");
    printf ("* Data: %02d.%02d.%02d.                    ",
        czas_akt->tm_mday, ((czas_akt->tm_mon) + 1), czas_akt->tm_year);
    printf ("                    Czas: %02d:%02d:%02d *\n",
        czas_akt->tm_hour, czas_akt->tm_min, czas_akt->tm_sec);
    printf ("****************************************");
    printf ("****************************************\n");
}
```

```
/* Obliczanie mnacierzy pozycji kolnierza robota IRp-6 na podstawie sinusow  *
 * i cosinusow katow w osiach - wykorzystanie przeksztalconych wzorow        */

#include <stdio.h>
#include <mem.h>
#include <math.h>
#include "imgmat.h"
#include "index.h"

int make_irp6_mac2 ( double *c, double *s, double (*avp)[4] )
{
  int i;

  avp[_1][_1] =  -c[_1] * s[_4] * c[_5] + s[_1] * s[_5];
  avp[_1][_2] =  c[_1] * s[_4] * s[_5] + s[_1] * c[_5];
  avp[_1][_3] =  c[_1] * c[_4];
  avp[_1][_4] =  c[_1] * (L[_3] * c[_3] - L[_2] * s[_2]);

  avp[_2][_1] =  -s[_1] * s[_4] * c[_5] - c[_1] * s[_5];
  avp[_2][_2] =  s[_1] * s[_4] * s[_5] - c[_1] * c[_5];
  avp[_2][_3] =  s[_1] * c[_4];
  avp[_2][_4] =  s[_1] * (L[_3] * c[_3] - L[_2] * s[_2]);

  avp[_3][_1] =  c[_4] * c[_5];
  avp[_3][_2] =  -c[_4] * s[_5];
  avp[_3][_3] =  s[_4];
  avp[_3][_4] =  L[_3] * s[_3] + L[_2] * c[_2];

  avp[_4][_1] = avp[_4][_2] = avp[_4][_3] = 0.;
  avp[_4][_4] = 1.;
/*printf ("Macierz A1*A2*A3*A4*A5 (wzory przeksztalcone):\n");
  for (i = 0; i < 4; i++)
    printf ("|  %8.2lf  |  %8.2lf  |  %8.2lf  |  %8.2lf  |\n",
            avp[i][0], avp[i][1], avp[i][2], avp[i][3]);
  printf ("\n"); */

  return ( 0 );
}
```

```
#include <stdio.h>
#include <math.h>
#include "index.h"

int rad_to_in_irp_6 ( poswew, c, s )
long poswew[];
double c[], s[];
{
  double x, y, z;
  double L2e, L3e;

    poswew[_1] = dtol(latan2(s[_1], c[_1]) / wsp1) + default_geom_pos[_1];

    poswew[_4] = dtol(latan2(s[_4], c[_4]) / wsp4) + default_geom_pos[_4];

    x = latan2(s[_5], c[_5]);
    y = wsp5 * (double)(poswew[_4] - default_geom_pos[_4] + default_geom_pos[_5]);
   if (x > y) x = x - 2 * M_PI;
    if (x <= (y - 2 * M_PI)) x = x + 2 * M_PI;
    poswew[_5] = -dtol(x / wsp5) + default_geom_pos[_5]
                 + poswew[_4] - default_geom_pos[_4];

    z = beta_2 - latan2(s[_2], c[_2]);
    y = cos(z);
    L2e = sqrt(LL2 - y * _2LL);
    poswew[_2] = dtol((L2e - L02) / c5_256) + default_geom_pos[_2];

    z = beta_3 - latan2(s[_3], c[_3]);
    y = cos(z);
    L3e = sqrt(LL2 - y * _2LL);
    poswew[_3] = dtol((L3e - L03) / c5_256) + default_geom_pos[_3];

  return ( 0 );
}
```

```
#include <stdio.h>
#include <aptypes.h>
#include <appublic.h>
#include <apgeopublic.h>
#include <uipublic.h>
#include <uierr.h>

/* Testing of interface between Borland C++ v. 4.0 and ROBCAD */

int k_user_inverse(component_name, desired_tcpf_location,
                   desired_tool_location, limits_flag,
                   location_mask, solutions, no_of_solution)

tName     component_name;
tFrame    desired_tcpf_location;
tFrame    desired_tool_location;
int       limits_flag;
int       location_mask;
tPoseArray  solutions;
int       no_of_solution;

{
double t_mat[][4], rpy[], tcp[], teta;

printf("test of k_user_inverse - start\n");

/* in the next lines you should put in correct names for components of
   desired_tcpf_location */
rpy[0] = desired_tcpf_location.roll;
rpy[1] = desired_tcpf_location.pitch;
rpy[2] = desired_tcpf_location.yaw;
tcp[0] = desired_tcpf_location.x;
tcp[1] = desired_tcpf_location.y;
tcp[2] = desired_tcpf_location.z;

/* Creating of full position matrix */
rpy_to_t_matrix (rpy, tcp, t_mat);
it_back_6 (t_mat, teta);

/* if solution is double array, it_back_6 can be called with
   solution as formal parameter */
for (i = 0; i < 5; i++) solutions[i] = teta[i];
no_of_solution = 1;

printf("test of k_user_inverse - exit\n");

return (1);
```