Appendix A

Modelling Real-time systems

The purpose of this appendix is to describe the tools of the modelling technique used for modelling the process control system in chapter 4. This description will delimited to the tools which are used to describe the process control system, and is based on [Ward et. al, vol. 1] and [Ward et. al, vol. 2]. The technique provides a language for describing the context, the architecture and the functionality of a real-time system. The technique is adapted from [Ward et al], in which the philosophy of this particular modelling technique is described in further details.

Defining the System Context

In order to understand the purpose and the functionality of a system, one have to become familiar with the role of the system. In this manner we have to identify the relations between the system boundaries and the surrounding environment. The context schema identifies the boundaries of the system, the in/output of the system and the external entities, by which the system interacts.

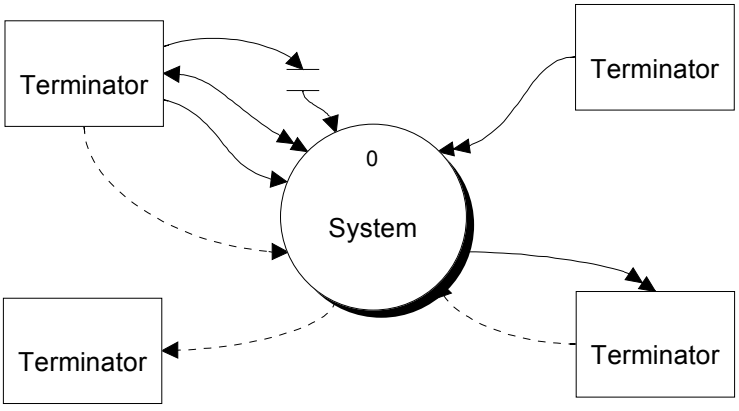


Figure A1: Context Schema notation [Ward et al, vol. 1. P. 15].

The system is represented by a single transformation (circle) which includes the entire functionality of the system. The external entities by which the system interacts are called terminators and is represented by boxes. The arrows describe the flows of data through the system interface and the direction of these flows.

Hierarchical composition

The transformation may be decomposed into a sub set of transformations. This decomposition serves the purpose of reaching a set of sub transformations, for which it becomes possible to get a thorough understanding of each transformation.

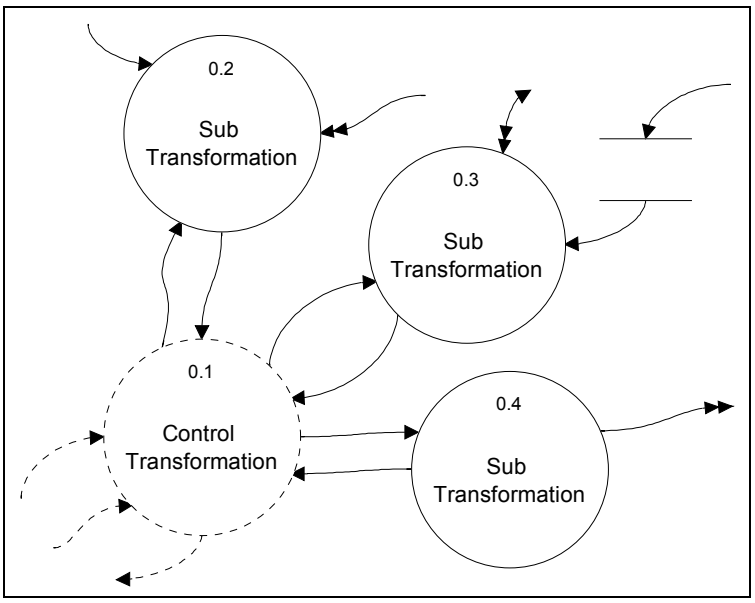


Figure A2: An example of decomposition. This figure illustrates a possible decomposition of the transformation in Figure A1.

When decomposing a transformation we must keep track of the flows coming in and going out. For a specific transformation these flows are preserved through the process of decomposing. In addition we will observe a number of flows between the new sub transformations. An example of this is shown in the transformation schema in Figure A2, which is a decomposition of the system transformation in the context diagram in Figure A1.

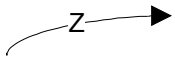
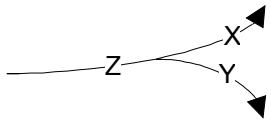
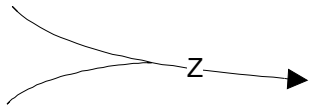
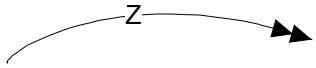
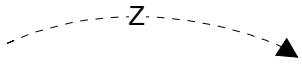
In order to facilitate recognition of the structure of the levels in the transformation schemas, we will number the transformations. The context schema will be numbered as 0, while the transformations of the next level of transformations will be numbered by a zero point the number of the specific sub transformation. When a transformation is decomposed into a lower level of transformations, it is indicated by a shadow, see Figure A3.

Handling Flows

As already observed we have more than one type of flows. The types of flows will be described in the following, and so will the storing of flows.

Conventions for Data Flows

This section serves as an overview of the different types of data flows. The flows are represented with their conventions in the following table.

	**Time-discrete data flow**. This type of flows will only contain data at individual points in time, and is considered to have an undefined value or null value at all other times.
	Two subsets of the time-discrete data flow Z are used by two different successor transformations
	All of the time-discrete data flow Z is used by two different successor transformations.
	The time-discrete data flow Z is composed of two different time-discrete data flows, X and Y. X and Y are provided from two different predecessors.
	All of the time-discrete data flow Z may be provided by either of two predecessor transformations
	Time-continuous data flow. This type of data flow exists at every instant within a time interval, such as a variable which value is a function of time.
	Event Flow. This type of flows has no content, i.e. it is only the presence of the flow which is interesting. An example is the signal from a button. The signal has no content, but only tells us that the button is pressed.

Storing Data

Sometimes we may need to store some data for later use. This can be done by use of two different storing mechanisms, the data store and the event store. The data store is usually served by transformations at discrete points in time. The flows between transformations and data stores will always be represented by single headed arrows. The data store is represented as shown in Figure A4.

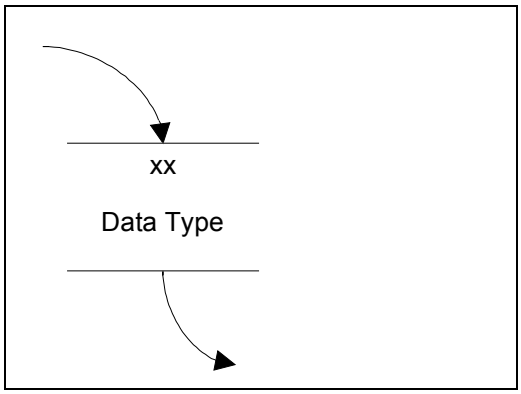


Figure A4: Data store.

The name of the data in the data store is the same as the name of the data flow which supplies the store with data, and which pulls out data from the store. Therefore the flows in and out of stores will not be labelled with data names, but the data in these flows are defined in the store.

The event store is analog to the data store, and it is represented by two parallel dotted lines like shown in

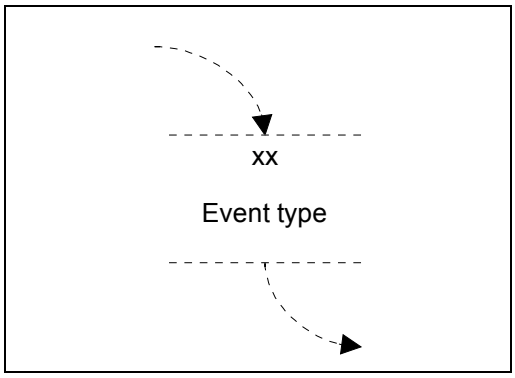


Figure A5: Event Store.

Types of Transformations

In the present modelling method we have two different types of transformations, the data transformation and the control transformation.

Data Transformation

The data transformation serves the function of transforming data from one format to another, i.e. performing calculations. The data transformations are visualised as a continuous full line circle as shown in

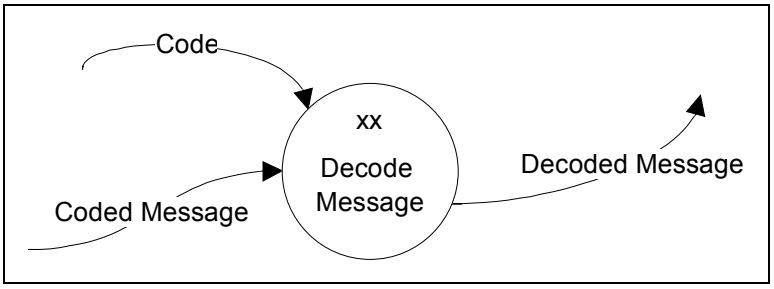


Figure A6: The data transformation serves the purpose of transforming coded messages into decoded messages [Ward et al, vol. 1. P. 42]

Control Transformation

A control transformation is represented by a dotted circle and serves only event flows like e.g. an on/off signal. This yields both for input and output signals. This type of Transformations is intended to control the sequence of data handling.

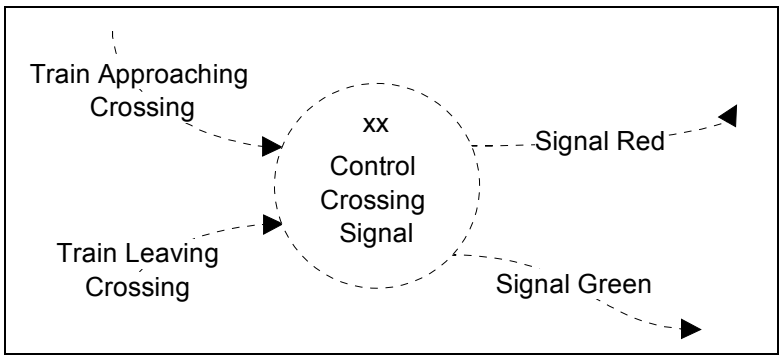


Figure A7: The control transformation is controlling the crossing signal. All flows are event flows. [Ward et al, vol. 1. P. 48]

In Figure A7 we can see an example of a simple control flow in which the crossing signal is controlled by the status of trains approaching and leaving the crossing.

Appendix B

Data Structures

This appendix states the definitions of data that is used in the functional architecture in chapter 5. The data is presented in alphabetical order.

1Calibration Parameters 2
Delta Equipment Time 2
Delta Sensor Time 2
Equipment Control Vector 2
Equipment Status 4
Estimated Workpiece Parameters and Positions for Flange joints 4
Estimated Workpiece Parameters and Positions for Flank joints 6
Extended Profile Nodes 7
Measured Profile Vector 8
Modelled Workpiece Parameters 10
Process Model Parameters 10
Profile Node 10

Profile Type 11
Robot Locations 11
Sensor Control Vector 11
Sensor Status 12
System Time 12
Transformed Profile Vectors 13
Welding Control Vectors 14

Calibration Parameters

The calibration parameters are the two 4 x 4-transformation matrices ${}^{sensor}T_{endeffector}$, which describes the transformation from the sensor frame to the endeffector frame and ${}^{TCP}T_{endeffector}$, which describes the transformation from TCP frame to the endeffector frame. The endeffector frame specifies the position and orientation of the endeffector of the mechanical robot structure.

Delta Equipment Time

The delta equipment time is a parameter, which is assigned the difference between the system time (value of timer in the welding control system) and the timer of the welding equipment controller. When the timer in the welding controller is reset, it will start counting from zero. At that moment the delta equipment time will be set to the actual value of the system time.

When the robot locations are received from the welding equipment controller, the welding equipment controller has defined the time stamps. When the delta equipment time is added to the time stamps, these are synchronised with the timer of the welding control system.

Delta Sensor Time

Delta sensor time has the same function as the delta equipment time, though it is used for synchronising the timestamps of the measured profile vectors with the time of the welding control system.

Equipment Control Vector

The equipment control vectors are the specific equipment commands. These must be specified in an equipment specific language. This description is general an specifies only the general types of commands.

The equipment control vector consists of a command part and a part for task specification for the equipment. The command part has the following possibilities:

- Set welding equipment in slave mode.
- Reset equipment time.
- Start sending robot locations.
- Movement Command.
- Reset welding equipment.

The task specification part is used for movement commands only. The contents is:

- Point type (End point or Not End point).
- Transformation matrix: ${}^{endeffector}T_{base}$.
- Velocity.
- Wire feed rate.

If the point type is ‘End point’ the execution of welding control vectors is stopped after execution of this equipment control vector.

Equipment Status

This response describes the status for execution of a specific equipment control vector. The following values of equipment status is possible:

- Accepted/Success/Fail.

If the command part of the equipment control vector is a *movement command* or *start sending robot locations*, the equipment status will respond with *accepted* or *fail*. This response only indicates if the equipment controller accepts the commands or not. The execution of these commands is not ended when the response is received. If the task specification part of a movement command is *end point*, the response is *success* or *fail*. If the response is *success*, the command is executed when the response is received.

If the command part of the equipment control vector is *reset equipment time* or *Set welding equipment in master mode*, the equipment status will respond with *success* or *fail*. Then the response is received after the command is executed, unless the execution fails.

Estimated Workpiece Parameters and Positions for Flange joints

The estimated workpiece parameters contain a local description of the weld groove. This description consists of geometric information as well as information about the workpiece conditions. The estimated workpiece parameters for Flange joints includes the following:

- Joint type
- Type of material for plate 1
- Type of material for plate 2
- Coating type, plate 1
- Coating type, plate 2
- Thickness of coating, plate 1
- Thickness of coating, plate 2
- Thickness of plate 1 (T1)

- Thickness of plate 2 (T2)
- Angle of workpiece 1 (α_1)
- Angle of workpiece 2 (α_2) (Only for flank-joints)
- Edgedist 1
- Edgedist 1
- Groove frame ($^{groove}T_{base}$)
- Gap
- Angle between surfaces (α_s)
- Tack/No tack
- Angle between welding direction and gravity
- Angle between weld bisector and gravity

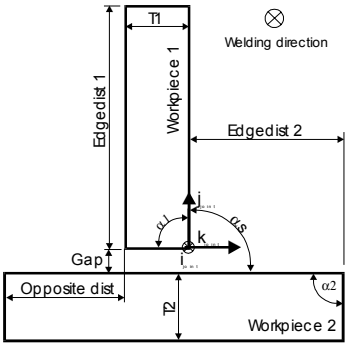


Figure 1: Geometric parameters in the estimated workpiece parameters for Flange-joints.

Estimated Workpiece Parameters and Positions for Flank joints

Flank joints are characterised by the position of the weld groove, which is between the ends of the workpieces. A flank joint is shown in figure 2 with the corresponding measured profile.

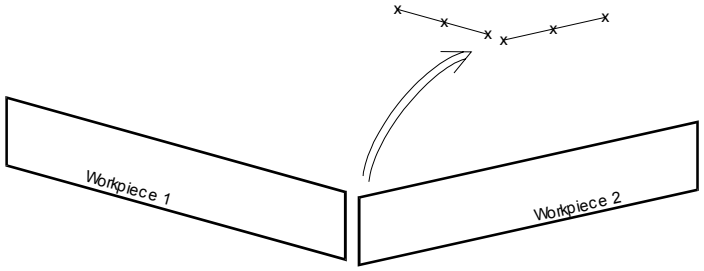


Figure 2: Flank joint and the corresponding measured profile.

The parameters mentioned in the section 'estimated workpiece parameters for Flange joints' are present in the estimated workpiece parameters for flank joints as well. Though three additional parameters are present in this structure:

- Off-set

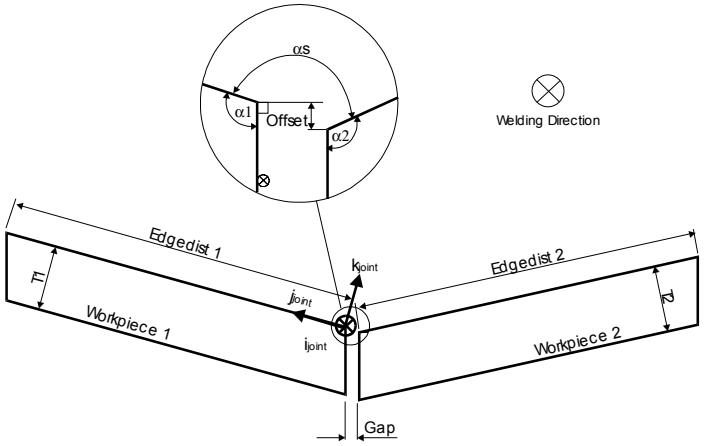


Figure 3: Flank joint and corresponding geometric parameters.

Extended Profile Nodes

An extended profile node is a local description of the weld groove by means of R, S and T points. The structure corresponds to the structure of profile nodes, except that the gap and the angle between the plates are included in the extended profile node.

The positions of R, S and T points are specified in the robot base frame. Additionally the profile node describes the fillet radius of workpiece 1 and 2. For Flange-joints the fillet radius is only specified for workpiece 1. The last three parameters of a profile node are booleans, which indicates whether or not one or both plates in the weld groove are missing at that particular position.

- R1
- S1
- T1
- R1
- S2
- T2
- Tack/No tack
- Gap
- Angle between workpiece surfaces
- Fillet radius (r_1)
- Fillet radius (r2) (Only for flank- and edge joints)
- Plate 1 missing
- Plate 2 missing
- Both plates missing

Measured Profile Vector

The measured profile vector includes a definition of three points on each workpiece in the weld groove (R, S and T) and a definition of the radius of fillets. The R, S and T – points defines the surface of a workpiece that may have a bevel shape at the weld groove.

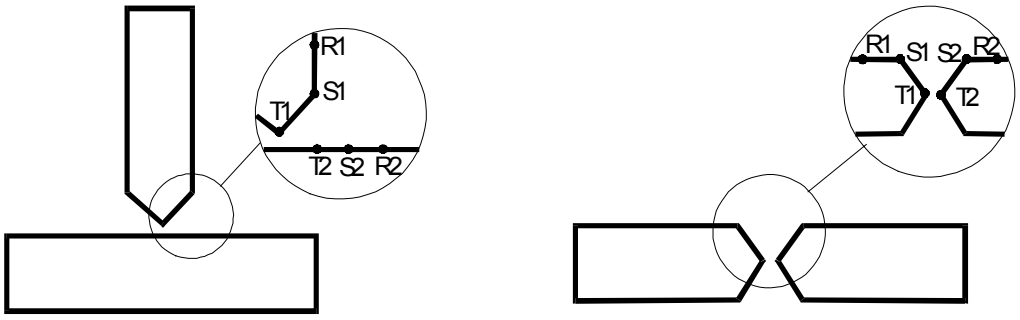


Figure 4: R, S and T points used for description of the weld groove profiles for T- and flank-joints when bevel shapes are present.

In case there is no bevel shapes in the weld groove it will be necessary to know an eventual fillet radius of the workpieces in order to recognize the position of the workieces and to calculate the size of the gap.

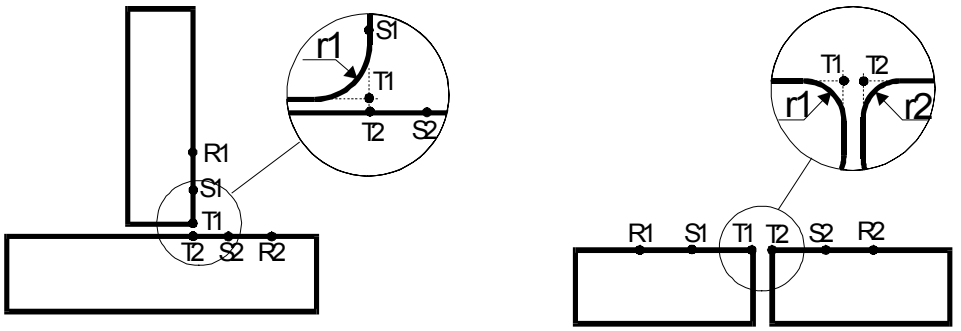


Figure 5: R, S and T points for description of the weld groove profiles for flange- and flank-joints when there are no bevel shapes.

Even though bevel shapes are normal in arc welding processes they are rarely used in laser welding. Though the preparation of workpieces may without intention cause the workpieces to have a small bevel or fillet radius. In case of bevel shape or fillet radius are present at the measuring position the points T1 and T2 are positioned as shown in Figure 4 and Figure 5. This

makes it possible to describe the exact workpiece geometry for these instances.

If a tack weld is present at the measured position the points T1 and T2 will be identical and describe the point of intersection between the lines described by |R1S1| and |R2S2| for T-joints. For flank joints the points T1 and T2 are also identical in the instance where a tack weld is recognised. These points are then positioned on the line that reaches the points R1 and R2. The point is the mid-point of the projection of the tack area on this line.

The measured profile vector includes a time stamp telling when the measurement is made in local sensor time.

The measured profile vector looks like this:

- Time Stamp
- R1
- S1
- T1
- R1
- S2
- T2
- Tack/No tack
- End/Not end
- Fillet radius (r_1)
- Fillet radius (r2) (Only for flank- and edge joints)
- Plate 1 missing
- Plate 2 missing
- Both plates missing
- Success/Fail

Modelled Workpiece Parameters

The modelled workpiece parameters specify the workpiece geometry, fit-up and surface treatment. The modelled workpiece parameters are:

Her indsættes statiske parametre fra workpiece model. Strukturering af forsk. fugetyper specificeres.

Process Model Parameters

The process model parameters are organised as a matrix. The first column of the matrix represents specific values of gap. The second third and fourth column represents a corresponding set of control variables. An inverse process model is shown below:

Gap	Welding speed	Wire feed rate	Displacement
0.0mm	1000mm/min	400mm/min	0.3mm
0.3mm	850mm/min	1800mm/min	0.4mm
0.6mm	500mm/min	3000mm/min	0.5mm
0.8mm	400mm/min	4000mm/min	0.7mm

Profile Node

A profile node is a local description of the weld groove by means of R, S and T points. The positions of R, S and T points are specified in the robot base frame. Additionally the profile node describes the fillet radius of workpiece 1 and 2. For Flange-joints the fillet radius is only specified for workpiece 1. The last three parameters of a profile node are booleans, which indicates whether or not one or both plates in the weld groove are missing at that particular position.

- R1
- S1
- T1
- R1
- S2
- T2

- Tack/No tack
- Fillet radius (r_1)
- Fillet radius (r2) (Only for flank- and edge joints)
- Plate 1 missing
- Plate 2 missing
- Both plates missing

Profile Type

The profile type specifies a set of generic geometric workpiece parameters of the weld groove. A number of profile types may be defined in the system. The profile types that are defined at this point is: Flank joint, Flange joint, edge-flank joints and edge-Flange joint.

Robot Locations

The robot controller continuously delivers information about the location of the robot. The robot locations consist of:

- Transformation matrix ($^{endeffector}T_{base}$).
- Time stamp.

Sensor Control Vector

The sensor control vector that consist of a command part and a task specification for the sensor. The command part must have the following possibilities:

- Set profile type
- Reset sensor time
- Start Profile Measurement
- Stop Measuring Profile
- The task specification part must have the following possibility:
- Profile type

The profile type may be specified by an integer, a string or similar. This specification serves as identification of the parameters that the sensor system must recognise and quantify in the measurements for a given profile type, such as a T-joint or a flank-joint.

For each profile type a number of characteristics are defined in the sensor system. The purpose of these characteristics is to facilitate precise measuring of the workpieces. These characteristics may not be known by the user of the sensor system and are therefore out of the scope of the present PhD-work.

Sensor Status

The sensor status is a vector that contains two logical values for the status of the requested operation and for measurement of a tack weld. The status for tackweld is set when the sensor system successfully has performed a measurement in which a tack weld is recognised. Additionally a value for the time at which the command is executed or failed in the sensor system.

The Sensor status looks like this:

- Success/Accepted/Not success
- Tack weld /No tack weld
- Time stamp

System Time

System Time is the actual timer value of the welding control system. This values changes continuously.

Transformed Profile Vectors

The transformed profile vectors corresponds to the measured profile vectors with the exception that the positions of R, S and T points are specified in the robot base frame. Additionally there is no time stamp in these vectors.

- R1
- S1
- T1
- R1
- S2
- T2
- Tack/No tack
- End/not end
- Fillet radius (r_1)
- Fillet radius (r2) (Only for flank- and edge joints)
- Plate 1 missing
- Plate 2 missing
- Both plates missing
- Success/Fail

Welding Control Vectors

The welding control vectors specifies the welding control variables for the welding process. These variables describes the motion of the laser beam by using the following variables:

TCP frame relative to world frame (${}^{TCP}T_{World}$)

Welding speed

Wire feed rate

Continuity type

The ${}^{TCP}T_{World}$ frame defines the position and orientation of the laser beam relative to the world frame. The origin of the TCP frame is placed in the focus point of the laser beam. The vector i_{TCP} points away from the welding equipment in the direction of the laser beam. ${}^{TCP}T_{World}$ is a transformation of ${}^{groove}T_{TCP}$.

The welding speed is the speed that is specified for the motion of TCP from the previous to the actual point.

The wire feed rate is the wire speed from the previous to the actual point.

The continuity type defines the type of point. The point may be a start point, an intermediate point or an end point. The execution process uses the continuity information. If e.g. the continuity is ‘end’ the execution will stop the execution will be stopped after executing that particular welding control vector.

Appendix C

Software

This appendix is a print of the software developed during the Ph.D. Project. The appendix is not explained or commented directly in the report, but is the implementation of chapter 4. Though the functional architecture in chapter 4 is updated with respect to gained experience during the work.

The software is implemented on a platform based on the operating system IRMX. The robot controller is implemented on the same platform. The robot controller is extended with "Compile Cycles" from Power Automation, Germany. By compile cycles it is possible to compile user specified routines programmed in the C language into the robot controller. The compile cycles system is event driven and calls the user programmed routines on specified events. The individual user written software modules are printed in the following with a very short introduction.

Module basic.ltx

This module defines the modules that are used during compilation of the software related to the basic compile cycle (see description later on this page).

Source code

```
czmain.obj,    &
GPIBdrv.obj,   &
SCAN.obj,      &
basic.obj,     &
matlib.obj,    &
queue.obj,     &
WlProCon.obj,  &
kalib.obj,
```

Module servo4.ltx

This module specifies the modules that are used during compilation of the software related to the servo compile cycle (see description under the module servo4.c).

Source code:

```
czmain.obj,                &
servo4.obj,                &
```

Module basic.c

This module communicates with the basic compile cycle in the robot controller. The basic compile cycle is a job in the IRMX control platform that controls the high level operations in the robot controller. The basic compile cycle generates events during these operations, and it is possible to interact with the robot controller by implementing user written procedures that are activated by the events of the basic job. Exchange of data is possible between the user written procedures and the robot controller. In this way NC-blocks is generated by the basic compile cycle and filled out by values, which are calculated in the user written procedures.

Souce code:

```
/*****************************************************************************
*
* Filename: basic.c
*
* Author: HJA
* Date  : 4. november 1996
*
* Date of last modification:
*
*****************************************************************************
*
* Specification:
* -------------

*/
#include <stdio.h>
#include <i86.h>
#include "cc_pa.h"
#include "WlProCon.h"
#include "Kalib.h"
#include "global.h"
#include "math.h"
#include "queue.h"


WORD_T      choice;
INT_T       status;
WORD_T      Old_NC_Status;

DWORD_T     Adr_Mask[26]  = {1, 2, 4, 8, 0x10, 0x20, 0x40, 0x80,
              0x100, 0x200, 0x400, 0x800, 0x1000, 0x2000,
              0x4000, 0x8000, 0x10000, 0x20000, 0x40000,
              0x80000, 0x100000, 0x200000, 0x400000, 0x800000,
              0x1000000, 0x2000000} ;

WORD_T      Bit_Mask[8]   = {1, 2, 4, 8, 0x10, 0x20, 0x40, 0x80} ;

I_NCBLK_T   inp1, inp2 ;
O_NCBLK_T   out1;
O_NCBLK_T   out2 ;

I_EVNT_T    inp5 ;
O_EVNT_T    out5 ;

I_COM_T    inpCom ;
O_COM_T    outCom ;

I_STAT_T    inpStat ;
O_STAT_T    outStat ;

#define Idle 0
#define MovingToStartPoint 1
#define Searching 2
#define Welding 3
#define WeldingFinished 4
#define FinishingNcProgram 5
#define ExecutionInterrupted 6
#define InitializingKalibration 7
#define PerformingKalibration 8
#define MovingToGrooveStartPoint 9
#define InitializingCoordinationIndexes 10
#define ReadyForInitialSCV 11
#define ExecutingSubroutine 12
#define Enabled  50
```

```
#define Disabled 51

unsigned int   BasicBlockTransfers;
unsigned int   BlockTransfersBeforeStart;
unsigned int   BlockTransfersBeforeEnd;
unsigned char  PA_STATUS;
unsigned char  OverWriteTemporaryLevel;
unsigned char  GenerateIntermediateBlock;
unsigned char  DisableBlockProcessing = FALSE;
unsigned char  EnableBlockProcessing = FALSE;
unsigned char  BlockStatus;
unsigned char  NewPositionFromServo = Disabled;
unsigned char  dummy;
unsigned char  TransferDummy;
unsigned char  PaFile = FALSE;
int  StartPointStrategy;
unsigned char  EndPointStrategy;
double T1, T2, Tstart;
int WelProConCommand;
char ExecuteSubroutine = FALSE;

/***************************
External function declarations:
-----------------------------*/
extern unsigned char KalibrationResponse;
extern unsigned char WelProConResponse;
extern void MessageText (char *TextBuf);

/***************************/


struct TransferredDataFromServo{
unsigned int   ServoBlockTransfers;
double ActualAxisPositions[5];
double TimeForPosition;
};

static struct TransferredDataFromServo ServoData;
extern struct ControlVector NewConVec;
static struct ControlVector MemActive, MemPassive, MemTemporary;
static struct ControlVector StartPoint, EndPoint;
FILE *pa;
FILE *axispos;

/*=============================================================================*/


void ResetApplicationProgram(void)
{
   if(PaFile)
   {
      fprintf(pa, "ResetApplicationProgram is executed\n");
      fflush (pa);
   }
   ResetWeldingProcessController();
   PA_STATUS = Idle;
   WelProConCommand = Idle;
   WelProConResponse = Idle;
   NewPositionFromServo = Disabled;
   BlockTransfersBeforeStart = 0;
   BlockTransfersBeforeEnd = 0;
   BlockStatus = Enabled;
   OverWriteTemporaryLevel = FALSE;
   GenerateIntermediateBlock = FALSE;
}
```

```
void ActivateEmergencyStop(void)
{
   I_STAT_T inpStat;
   O_STAT_T outStat;

   choice = 10 ;            /* Activate Emergency Stop */
   (*b_stat) (choice, &status, &inpStat, &outStat) ;
}

void CheckStatus(void)
{
   #define  Restart        0x0001
   #define  EmergencyOff   0x0002
   #define  CycleOff       0x0004
   #define  CycleOn        0x0008
   #define  CycleStop      0x0010
   #define  ProgramHalt    0x0020
   #define  ProgramEnd     0x0040
   #define  DwellTime      0x0080
   #define  ControlReset   0x0100
   #define  HomingTravel   0x0200
   #define  Buffering      0x0400

   I_STAT_T inpStat;
   O_STAT_T outStat;
   WORD_T NC_Status;

   choice = 1 ;             /* read NC status */
   (*b_stat) (choice, &status, &inpStat, &outStat) ;
   if (status == 0)
   {
      NC_Status = 0;
      NC_Status = (NC_Status | outStat.O1_STAT.NCSTATUS);
      if  ((NC_Status & EmergencyOff) || (NC_Status & ControlReset)
           /* || (NC_Status & Restart) || (NC_Status & ProgramHalt)*/)
      {
         if(PaFile)
         {
            fprintf(pa, "NC Status = %d\n", NC_Status);
            fflush(pa);
         }
         ResetApplicationProgram();
         BasicBlockTransfers = ServoData.ServoBlockTransfers;
      }
   }
}

void OverWriteTemporaryLevelWithStartPoint(void)
{
   NewConVec.AxisPosition[0] = StartPoint.AxisPosition[0];  /* Axis B */
   NewConVec.AxisPosition[1] = StartPoint.AxisPosition[1];  /* Axis B */
   NewConVec.AxisPosition[2] = StartPoint.AxisPosition[2];  /* Axis C */
   NewConVec.AxisPosition[5] = StartPoint.AxisPosition[5];  /* Velocity */
   NewConVec.AxisPosition[6] = 9;  /* G-word; Activate look ahead function*/
   NewConVec.AxisPosition[18] = 1000; /* Wire Feed Speed */    /* Axis S */
   NewConVec.AxisPosition[20] = 0; /* Voltage Output for Laser Power */
   NewConVec.AxisPosition[23] = StartPoint.AxisPosition[23];/* Axis X */
   NewConVec.AxisPosition[24] = StartPoint.AxisPosition[24];/* Axis Y */
   NewConVec.AxisPosition[25] = StartPoint.AxisPosition[25];/* Axis Z */
   OverWriteTemporaryLevel = TRUE;
   if(PaFile)
   {
      fprintf(pa, "StartPoint is copied into NewConVec\n");
      fflush(pa);
   }
}
```

```
double GetSystemTime(void)
{
    double SystemSeconds;
    double SystemMinutes;
     choice = 15;
     (*b_stat) (choice, &status, &inpStat, &outStat);
     SystemSeconds = outStat.O15_STAT.SYSTEMUHR;
     SystemMinutes = outStat.O15_STAT.SYSTEMUHRMIN;
     SystemSeconds = SystemSeconds/1000 + SystemMinutes * 60;
     return SystemSeconds;
}

void far main(unsigned short event)
{
    switch (event)
    {

    /*------------------ Basis ----------------------------------------*/
       case E_START:
       {
          choice = 1;             /* set events */

          inp5.I1_EVNT.EVENT = E_B4000;
          (*b_evnt) (choice, &status,  &inp5, &out5);

          inp5.I1_EVNT.EVENT = E_INTPRET_END;
          (*b_evnt) (choice, &status,  &inp5, &out5);

          inp5.I1_EVNT.EVENT = E_BASIS_SERVO_COM;
          (*b_evnt) (choice, &status,  &inp5, &out5);

          inp5.I1_EVNT.EVENT = E_GCODE_START;
          (*b_evnt) (choice, &status,  &inp5, &out5);

          inp5.I1_EVNT.EVENT = E_TRANSFER;
          (*b_evnt) (choice, &status,  &inp5, &out5);

          if(PaFile)
          {
             pa = fopen("PA.LOG","w+");
             fprintf(pa, "PA.LOG IS OPENED");
             fflush(pa);
          }
       }
       break;  /** End of "case E_START:" **/

    /*------------------Basis-----------------------------------------*/
       case E_B4000:
       {
          if((BasicBlockTransfers > 0) && (PA_STATUS != Idle))
           CheckStatus();
       /**********************************/
          if((NewPositionFromServo == TRUE) && (PA_STATUS != Idle))
           {
             tryUpdateRobotLoc((ServoData.TimeForPosition - Tstart),
ServoData.ActualAxisPositions[0],
                ServoData.ActualAxisPositions[1],ServoData.ActualAxisPositions[2],
                ServoData.ActualAxisPositions[4], ServoData.ActualAxisPositions[3]);
             NewPositionFromServo = FALSE;
             if(PaFile)
             {
                fprintf(pa, "ServoData.TimeForPosition:
%f\n",ServoData.TimeForPosition);
                fflush (pa);
             }
          }
```

```
/*********************************/
   if ((PA_STATUS == InitializingKalibration) &&
       (BasicBlockTransfers == BlockTransfersBeforeStart) &&
       (BlockStatus != Disabled))
   {
       DisableBlockProcessing = TRUE;
       if(PaFile)
       {
          fprintf(pa, "PA_STATUS = InitializingKalibration");
          fflush(pa);
       }
   }
/*********************************/
   else if((PA_STATUS == InitializingKalibration) &&
          (ServoData.ServoBlockTransfers == BlockTransfersBeforeStart)&&
          (ServoData.ActualAxisPositions[3] == StartPoint.AxisPosition[2]) &&
          (ServoData.ActualAxisPositions[4] == StartPoint.AxisPosition[1]) &&
          (ServoData.ActualAxisPositions[0] == StartPoint.AxisPosition[23]) &&
          (ServoData.ActualAxisPositions[1] == StartPoint.AxisPosition[24]) &&
          (ServoData.ActualAxisPositions[2] == StartPoint.AxisPosition[25]) &&
          (BlockStatus == Disabled))
   {
       WelProConCommand = InitializeKalibration();
       if (KalibrationResponse == KalibrationInitialized)
       {
          PA_STATUS = PerformingKalibration;
          if(PaFile)
          {
             pa = fopen("pa.log", "w+");
             fprintf(pa, "KalibrationInitialized");
             fprintf(pa, "PA_STATUS is set to PerformingKalibration");
             fprintf(pa, "KalibrationResponse = KalibrationInitialized");
             fprintf(pa, "PA_STATUS is set to PerformingKalibration");
             fflush(pa);
          }
       }
   }
 /*********************************/
   else if (PA_STATUS == PerformingKalibration)
   {
       WelProConCommand = PerformKalibration();
       if(PaFile)
       {
          fprintf(pa, "PerformKalibration is called");
          fflush(pa);
       }
       if (KalibrationResponse == KalibrationFinished)
       {
          PA_STATUS = Idle;
          EnableBlockProcessing = TRUE;
          if (PaFile)
          {
             fprintf(pa,"KalibrationResponse = KalibrationFinished");
             fflush(pa);
          }
       }
   }
 /*********************************/
   else if (PA_STATUS == WeldingFinished)
      ResetApplicationProgram();
 /*********************************/
   else if((PA_STATUS == MovingToStartPoint) &&
       (BasicBlockTransfers < BlockTransfersBeforeStart) && (TransferDummy))
   {
       OverWriteTemporaryLevelWithStartPoint();
   }
 /*********************************/
```

```
          else if ((PA_STATUS == MovingToStartPoint) &&
              (BasicBlockTransfers == BlockTransfersBeforeStart) &&
              (BlockStatus != Disabled))
          {
             DisableBlockProcessing = TRUE;
          }
          /******** Start of PA_STATUS == MovingToStartPoint ********/

          else if ((PA_STATUS == MovingToStartPoint) &&
            /* (ServoData.ServoBlockTransfers == BlockTransfersBeforeStart) &&*/
              (sqrt(square(ServoData.ActualAxisPositions[3] -
StartPoint.AxisPosition[2]))<10) &&
              (sqrt(square(ServoData.ActualAxisPositions[4] -
StartPoint.AxisPosition[1]))<10) &&
              (sqrt(square(ServoData.ActualAxisPositions[0] -
StartPoint.AxisPosition[23]))<10) &&
              (sqrt(square(ServoData.ActualAxisPositions[1] -
StartPoint.AxisPosition[24]))<10) &&
              (sqrt(square(ServoData.ActualAxisPositions[2] -
StartPoint.AxisPosition[25]))<10) &&
              (BlockStatus == Disabled))
          {
             if(PaFile)
             {
                fprintf(pa,"Start Point is Reached\n");
                fflush(pa);
             }

          /****** Start Set Template And Syncronize Time ******/

             WelProConCommand = SetSensorTemplate(ServoData.ActualAxisPositions[3]);
             if (WelProConResponse == SensorTemplateSet)
             {
                T1 = GetSystemTime();
                WelProConCommand = SyncronizeSensorTime();
                if (WelProConResponse == SensorTimeSyncronized)
                {
                   T2 = GetSystemTime();
                   Tstart = (T2-T1)/2 + T1;
                   NewPositionFromServo = Enabled;
                   if (PaFile)
                   {
                      fprintf(pa,"System- And SensorTimeSyncronized\n");
                      fprintf(pa,"T2 - T1 = %f\n",(T2-T1));
                      fflush (pa);
                   }
                }
             }
          /****** End Set Template And Syncronize Time ******/

          /********** Start Initialize Sensor **************/

             if (WelProConResponse == SensorTimeSyncronized)
                WelProConCommand = InitializeSensor();
             if (WelProConResponse == SensorInitialized)
             {
                PA_STATUS = InitializingCoordinationIndexes;
                if (PaFile)
                {
                   fprintf(pa,"SensorInitialized\n");
                   fprintf(pa,"PA_STATUS = InitializingCoordinationIndexes\n");
                   fflush (pa);
                }
             }
          /********** End Initialize Sensor **************/

          } /** "End of PA_STATUS == MovingToStartPoint" **/
```

```
   else if (PA_STATUS == InitializingCoordinationIndexes)
   {
      WelProConCommand = InitializeRstIndex();
      if (WelProConResponse == InitializingRstIndex)
         if(PaFile)
         {
            fprintf(pa, "InitializingRstIndex");
            fflush(pa);
         }
      if (WelProConResponse == InitializingRstIndexFinished)
      {
         PA_STATUS = ReadyForInitialSCV;
         if(PaFile)
         {
            fprintf(pa, "InitializingRstIndex");
            fflush(pa);
         }
      }
   }
   /******** Start Make Initial Control Vector *******/
   if (PA_STATUS == ReadyForInitialSCV)
   {
      WelProConCommand = GetInitSCV();
      if(WelProConResponse == InitSCVGenerated)
      {
         PA_STATUS = MovingToGrooveStartPoint;
         if((BasicBlockTransfers >= BlockTransfersBeforeEnd) &&
            (BlockTransfersBeforeEnd > 0))
            GenerateIntermediateBlock = TRUE;
         if (BlockStatus == Disabled)
            EnableBlockProcessing = TRUE;
         OverWriteTemporaryLevel = TRUE;
         if(PaFile)
         {
            fprintf(pa,"Initial SCV is Generated\n");
            fflush(pa);
         }
      }
/******** End Make Initial Control Vector *******/
   }
/******** End of PA_STATUS == ReadyForInitialSCV ********/

   else if ((PA_STATUS == MovingToGrooveStartPoint) && (TransferDummy))
   {
      if(PaFile)
      {
         fprintf(pa,"PA_STATUS = MovingToGrooveStartPoint\n");
         fflush(pa);
      }
      WelProConCommand = GetSuccSCV(ServoData.ActualAxisPositions[0],
         ServoData.ActualAxisPositions[1],ServoData.ActualAxisPositions[2],
         ServoData.ActualAxisPositions[4], ServoData.ActualAxisPositions[3]);
      if (PaFile)
      {
         fprintf(pa,"WelProConResponse = %d ",WelProConResponse);
         fflush(pa);
      }
      if(WelProConResponse == SuccSCVGenerated)
      {
         PA_STATUS = Searching;
         if((BasicBlockTransfers >= BlockTransfersBeforeEnd) &&
            (BlockTransfersBeforeEnd > 0))
            GenerateIntermediateBlock = TRUE;
         if (BlockStatus == Disabled)
            EnableBlockProcessing = TRUE;
            OverWriteTemporaryLevel = TRUE;
```

```
            if(PaFile)
            {
                fprintf(pa,"Second SCV is Generated\n");
                fflush(pa);
            }
        }
        else if (TransferDummy)
            DisableBlockProcessing = TRUE;
    }
    else if (PA_STATUS == Searching)
    {
        if(PaFile && TransferDummy)
        {
            fprintf(pa, "PA_STATUS = Searching\n");
            fflush(pa);
        }
        WelProConCommand = GetSuccSCV(ServoData.ActualAxisPositions[0],
            ServoData.ActualAxisPositions[1],ServoData.ActualAxisPositions[2],
            ServoData.ActualAxisPositions[4], ServoData.ActualAxisPositions[3]);
        if(WelProConResponse == SuccSCVGenerated)
        {
            if(PaFile)
            {
                fprintf(pa, "Succeeding SCV Generated\n");
                fflush(pa);
            }
            if((BasicBlockTransfers >= BlockTransfersBeforeEnd) &&
                (BlockTransfersBeforeEnd > 0))
                GenerateIntermediateBlock = TRUE;
            OverWriteTemporaryLevel = TRUE;
            if (BlockStatus == Disabled)
            EnableBlockProcessing = TRUE;
        }
        else if (WelProConResponse == StartPointDefined)
        {
            PA_STATUS = FinishingNcProgram;
            EnableBlockProcessing = TRUE;
            ExecuteSubroutine = TRUE;
            OverWriteTemporaryLevel = TRUE;

            if(PaFile)
            {
                fprintf(pa, "PA_STATUS = FinishingNcProgram\n");
                fflush(pa);
            }
        }
        else if (TransferDummy)
            DisableBlockProcessing = TRUE;
    }
    /*************** End of Searching *****************/

    else if (PA_STATUS == Welding)
    {
        if (ExecuteSubroutine == FinishingNcProgram)
        {
            DisableBlockProcessing = TRUE;
            ExecuteSubroutine = FALSE;
            GenerateIntermediateBlock = TRUE;
        }
        if(PaFile)
        {
            fprintf(pa, "PA_STATUS = Welding\n");
            fflush(pa);
        }
```

```
            WelProConCommand = GetWCV(ServoData.ActualAxisPositions[0],
                ServoData.ActualAxisPositions[1],ServoData.ActualAxisPositions[2],
                ServoData.ActualAxisPositions[4], ServoData.ActualAxisPositions[3]);
            if (WelProConResponse == WCVGenerated)
            {
                if((BasicBlockTransfers >= BlockTransfersBeforeEnd) &&
(BlockTransfersBeforeEnd > 0))
                    GenerateIntermediateBlock = TRUE;
                OverWriteTemporaryLevel = TRUE;
                if (BlockStatus == Disabled)
                    EnableBlockProcessing = TRUE;
                if(PaFile)
                {
                    fprintf(pa,"WelProConResponse = WCVGenerated\n");
                    fflush(pa);
                }
            }
            else if (WelProConResponse == EndOfWCVQueue)
            {
                PA_STATUS = WeldingFinished;
                NewPositionFromServo = Disabled;
                if((BasicBlockTransfers >= BlockTransfersBeforeEnd) &&
                    (BlockTransfersBeforeEnd > 0))
                    GenerateIntermediateBlock = TRUE;
                OverWriteTemporaryLevel = TRUE;
                if (BlockStatus == Disabled)
                    EnableBlockProcessing = TRUE;
                if(PaFile)
                {
                    fprintf(pa,"Final Control Vector is Generated\n");
                    fflush(pa);
                }
            }
            else if (TransferDummy == TRUE)
                DisableBlockProcessing = TRUE;
        }
        else if ((PA_STATUS == WeldingFinished)&&(BlockStatus == Disabled))
            EnableBlockProcessing = TRUE;

        /*** Enable Block Processing Start ***/
        if((EnableBlockProcessing == TRUE) && (BlockStatus != Enabled))
        {
            choice = 6;
            inp2.I6_NCBLK.NC_STATUS = Old_NC_Status;
            (*b_ncblk) (choice, &status, &inp2, &out2) ;
            EnableBlockProcessing = FALSE;
            BlockStatus = Enabled;
            if(PaFile)
            {
                fprintf(pa, "EnableBlockProcessing Performed\n");
                fflush(pa);
            }
        }
        /*** Enable Block Processing End ***/

        if (DisableBlockProcessing && (BlockStatus != Disabled))
        {
            /****   Read Old NC_status   ****/
            choice = 5;
            (*b_ncblk) (choice, &status, &inp2, &out2) ;
            Old_NC_Status = out2.O5_NCBLK.NC_STATUS;
            if(PaFile)
            {
                fprintf(pa,"Old_NC_status read = %d\n",Old_NC_Status);
                fflush(pa);
            }
```

```
            /**** Disable Block Processing ****/
            choice = 6;
            inp2.I6_NCBLK.NC_STATUS = 14;
            (*b_ncblk) (choice, &status, &inp2, &out2) ;
            BlockStatus = Disabled;
            DisableBlockProcessing = FALSE;
            if(PaFile)
            {
               fprintf(pa,"Blockprocessing Disabled\n");
               fflush(pa);
            }
         }
      }
      TransferDummy = FALSE;
      break;  /** End of "case E_B4000:" **/

   /*------------------Basis-----------------------------------------*/

      case E_INTPRET_END:
      {
      /**************** Reading Temporary Level Start ***************/
         choice = 3 ;
         inp1.I3_NCBLK.LEVEL = 0 ;
         (*b_ncblk) (choice, &status, &inp1, &out1) ;
         for (dummy = 0; dummy < 26; dummy++)
         {
            MemTemporary.AxisPosition[dummy] = out1.O3_NCBLK.NC_ADDRESS[dummy];
         }
         if(PaFile)
         {
            fprintf(pa, "Temporary Level is read\n");
            fflush(pa);
         }
      /**************** Reading Temporary Level End *****************/

      /************ Start Determine State of NC Program *************/
         if ((MemTemporary.AxisPosition[6] - 400 >= 0) &&
             (MemTemporary.AxisPosition[6] - 400 < 100))
         {
            StartPointStrategy = MemTemporary.AxisPosition[6];
            StartPoint.AxisPosition[0]  = MemTemporary.AxisPosition[0];
            StartPoint.AxisPosition[1]  = MemTemporary.AxisPosition[1];
            StartPoint.AxisPosition[2]  = MemTemporary.AxisPosition[2];
            StartPoint.AxisPosition[5]  = MemTemporary.AxisPosition[5];
            StartPoint.AxisPosition[6]  = 9;  /* G Code */
         /* StartPoint.AxisPosition[12] = 03; /* Start wire feed spindel */
            StartPoint.AxisPosition[18] = 0;  /* Wire Feed Rate */
            StartPoint.AxisPosition[20] = 0;  /* Laser Power */
            StartPoint.AxisPosition[23] = MemTemporary.AxisPosition[23];
            StartPoint.AxisPosition[24] = MemTemporary.AxisPosition[24];
            StartPoint.AxisPosition[25] = MemTemporary.AxisPosition[25];
            PA_STATUS = MovingToStartPoint;
            BlockTransfersBeforeStart = BasicBlockTransfers + 2;
            chooseIpm(StartPointStrategy - 400);
            if(PaFile)
            {
               fprintf(pa, "PA_STATUS = MovingToStartPoint\n");
               fprintf(pa, "BlockTransfersBeforeStart =
%d\n",BlockTransfersBeforeStart);
               fprintf(pa, "BasicBlockTransfers: %d\n", BasicBlockTransfers);
               fprintf(pa, "ServoBlockTransfers: %d\n",
ServoData.ServoBlockTransfers);
               fflush(pa);
            }
         }
```

```
      if ((MemTemporary.AxisPosition[6] - 500 >= 0) &&
          (MemTemporary.AxisPosition[6] - 500 < 100))
         {
            EndPointStrategy = MemTemporary.AxisPosition[6];
            BlockTransfersBeforeEnd = BasicBlockTransfers;
            GenerateIntermediateBlock = TRUE;
            if(PaFile)
            {
               fprintf(pa, "PA_STATUS = G500 is found\n");
               fflush(pa);
            }
         }
         if (MemTemporary.AxisPosition[6] == 399)
         {
            PA_STATUS = InitializingKalibration;

            BlockTransfersBeforeStart = BasicBlockTransfers + 2;
            StartPoint.AxisPosition[0]  = MemTemporary.AxisPosition[0];
            StartPoint.AxisPosition[1]  = MemTemporary.AxisPosition[1];
            StartPoint.AxisPosition[2]  = MemTemporary.AxisPosition[2];
            StartPoint.AxisPosition[23] = MemTemporary.AxisPosition[23];
            StartPoint.AxisPosition[24] = MemTemporary.AxisPosition[24];
            StartPoint.AxisPosition[25] = MemTemporary.AxisPosition[25];

            if(PaFile)
            {
               fprintf(pa, "PA_STATUS = InitializingKalibration\n");
               fflush(pa);
            }
         }

         if ((MemTemporary.AxisPosition[12] == 30) && (ExecuteSubroutine ==
FinishingNcProgram))
         {

            PA_STATUS = Welding;
            if(PaFile)
            {
               fprintf(pa, "PA_STATUS is Set to Welding\n");
               fflush(pa);
            }
         }

      /************* End Determine State of NC Program **************/
      }
      break; /** "case E_INTPRET_END:" **/

   /*------------------Basis-------------------------------------------*/

      case E_GCODE_START:
      {

      /************ Overwrite Temporary Level Start ***********/
         if (OverWriteTemporaryLevel == TRUE)
         {
            choice = 4 ;
            inp1.I4_NCBLK.LEVEL = 0 ;
            inp1.I4_NCBLK.NC_PROG = 1 ;
            for (dummy = 0; dummy < 26; dummy++)
            {
               if (dummy == 23 || dummy == 24 || dummy == 25)
               {
                  inp1.I4_NCBLK.NC_ADDRESS = dummy ;
                  inp1.I4_NCBLK.NC_VALUE = NewConVec.AxisPosition[dummy];
                  (*b_ncblk) (choice, &status, &inp1, &out1) ;
               }
            }
```

```
            if (PA_STATUS == Welding || PA_STATUS == WeldingFinished)
            {
                  inp1.I4_NCBLK.NC_ADDRESS = 5;
                  inp1.I4_NCBLK.NC_VALUE = NewConVec.AxisPosition[5];
                  (*b_ncblk) (choice, &status, &inp1, &out1) ;

                  inp1.I4_NCBLK.NC_ADDRESS = 6;
                  inp1.I4_NCBLK.NC_VALUE = 1;
                  (*b_ncblk) (choice, &status, &inp1, &out1) ;

                  inp1.I4_NCBLK.NC_ADDRESS = 18;
                  inp1.I4_NCBLK.NC_VALUE = NewConVec.AxisPosition[18];
                  (*b_ncblk) (choice, &status, &inp1, &out1) ;
            }
            OverWriteTemporaryLevel = FALSE;
            if(PaFile)
            {
               pa = fopen("pa.log","a+");
               fprintf(pa, "Temporary Level is Overwritten\n");
               fprintf(pa,"NewConVec[5] = %f\n",NewConVec.AxisPosition[5]);
               fprintf(pa,"NewConVec[18] = %f\n",NewConVec.AxisPosition[18]);
               fprintf(pa,"PA.LOG IS CLOSED");
               fclose(pa);
            }
         } /** "if OverWriteTemporaryLevel == TRUE" **/


         if (ExecuteSubroutine == TRUE)
         {
            choice = 4 ;
            inp1.I4_NCBLK.LEVEL = 0 ;
            inp1.I4_NCBLK.NC_PROG = 1 ;
            if(StartPointStrategy == 401) /* G401 = kalibrering */
            {
               inp1.I4_NCBLK.NC_ADDRESS = 6 ; /* G-kode vaelges */
               inp1.I4_NCBLK.NC_VALUE = 1; /* vaerdi saettes til 4
(forsinkelsesblok)*/
            }
            else
            {
               inp1.I4_NCBLK.NC_ADDRESS = 16 ; /* Q-kode vaelges (kald underprogram)
*/
               inp1.I4_NCBLK.NC_VALUE = 222;   /* vaerdi saettes til 222 (program nr)
*/
            }
            (*b_ncblk) (choice, &status, &inp1, &out1) ;

            ExecuteSubroutine = FinishingNcProgram;
            OverWriteTemporaryLevel = FALSE;
            if(PaFile)
            {
               fprintf(pa, "Temporary Level is Overwritten\n");
               fprintf(pa,"Subroutine is started\n");
               fflush(pa);
            }
         } /** "if OverWriteTemporaryLevel == TRUE" **/

      /************* Overwrite Temporary Level End ************/

      /********** Generate Intermediate Block Start ***********/
         if (GenerateIntermediateBlock == TRUE)
         {
            choice = 9;
            inp1.I9_NCBLK.INTERPOSE_FLAG = 2;
            (*b_ncblk) (choice, &status, &inp1, &out1) ;
            GenerateIntermediateBlock = FALSE;
```

```
            if(PaFile)
                {
                    fprintf(pa, "Intermediate Block is Generated\n");
                    fflush(pa);
                }
            }  /**"if GenerateIntermediateBlock == TRUE" **/
        /*********** Generate Intermediate Block End ************/
        }
        break;   /** End of "case E_GCODE_START:" **/

    /*------------------Basis-----------------------------------------*/


        case E_TRANSFER:
        {
            MemActive    = MemPassive;
            MemPassive   = MemTemporary;
            BasicBlockTransfers++;
            TransferDummy = TRUE;
            if(PaFile)
            {
                fprintf(pa, "BasicBlockTransfers = %d\n",BasicBlockTransfers);
                fprintf(pa, "ServoBlockTransfers = %d\n",ServoData.ServoBlockTransfers);
                fflush(pa);
            }
        }
        break;  /** "case E_TRANSFER:" **/


         /*------------------Basis-----------------------------------------*/

        case E_BASIS_SERVO_COM:
        {
        /**** Recieve Servodata from Servo Process Start *****/

            choice = 22;
            inpCom.I22_COM.OFFSET = 0;
            inpCom.I22_COM.LEN = sizeof(ServoData);
            inpCom.I22_COM.DEST_22 = &ServoData;
            (*b_com)(choice, &status, &inpCom, &outCom);
            if((NewPositionFromServo == FALSE) || (NewPositionFromServo == Enabled))
                NewPositionFromServo = TRUE;
        /****** Recieve Servodata from Servo Process End ******/
        }
        break;  /** End of "case E_BASIS_SERVO_COM:" **/

    }  /** "End switch (event)" **/
} /** "void far main(unsigned short event)" **/
```

Module servo4.c

This module communicates with the servo compile cycle in the robot controller. The servo compile cycle is a job in the IRMX control platform that controls the lower level operations in the robot controller, such as control of the servo drives, input, output etc. The servo compile cycle generates events during these operations, and it is possible to interact with the robot controller by implementing user written procedures that are activated by the events of the servo job. Exchange of data is possible between the user written procedures and the robot controller. In this way

The communication with the servo job is used for reading immediate positions of the robot axes and to transfer these values to the module basic.c.

Source code:

```
/*****************************************************************************
*
* Filename: servo4.c
*
* Author: HJA
* Date  : 4. october 1996
*
* Date of last modification:
*
*****************************************************************************
*
* Specification:
* -------------
*/

#include <stdio.h>
#include <i86.h>
#include <cc_pa.h>

WORD_T      choice;
INT_T       status;
WORD_T      Old_NC_Status;
float MinuteCounter = 0;
float PreviousClock = 0;
float DummyClock;
DWORD_T     Adr_Mask[26]  = {1, 2, 4, 8, 0x10, 0x20, 0x40, 0x80,
0x100, 0x200, 0x400, 0x800, 0x1000, 0x2000,
0x4000, 0x8000, 0x10000, 0x20000, 0x40000,
0x80000, 0x100000, 0x200000, 0x400000, 0x800000,
0x1000000, 0x2000000} ;

WORD_T      Bit_Mask[8]   = {1, 2, 4, 8, 0x10, 0x20, 0x40, 0x80} ;

struct ExecutedData {
   unsigned int   NumberOfServoBlocksTransferred;
   double  AxisPos[5];
   double  ServoClock;
   };
struct ExecutedData CommunicationToBasic;

I_COM_T     inpCom;
O_COM_T     outCom;

I_AXES_T    inpAxes;
O_AXES_T    outAxes;

I_EVNT_T    inpEvnt;
O_EVNT_T    outEvnt;

I_STAT_T    inpStat;
O_STAT_T    outStat;


/*===========================================================================*/
```

```
void far main(unsigned short event)
{
   switch (event)
      {
      /*------------------Servo-----------------------------------------*/
      case E_START:
      {
         choice = 1;             /* set events */

         inpEvnt.I1_EVNT.EVENT = E_TRANSFER;
         (*b_evnt) (choice, &status,  &inpEvnt, &outEvnt);

         inpEvnt.I1_EVNT.EVENT = E_SERVO_BASIS_COM;
         (*b_evnt) (choice, &status,  &inpEvnt, &outEvnt);

         inpEvnt.I1_EVNT.EVENT = E_POSPROC_VALUES;
         (*b_evnt) (choice, &status,  &inpEvnt, &outEvnt);

         CommunicationToBasic.NumberOfServoBlocksTransferred = 0;
      }
      break;
      /*------------------Servo-----------------------------------------*/
      case E_SERVO_BASIS_COM:
      {
         choice = 24;
         inpCom.I24_COM.OFFSET = 0;
         inpCom.I24_COM.LEN = sizeof(CommunicationToBasic);
         inpCom.I24_COM.SOURCE_24 = &CommunicationToBasic;
         (*b_com) (choice, &status, &inpCom, &outCom);
      }
      break;
       /*------------------Servo-----------------------------------------*/

      case E_POSPROC_VALUES:
      {
         choice = 15;
         (*b_stat) (choice, &status, &inpStat, &outStat);
         DummyClock = outStat.O15_STAT.SYSTEMUHR;
         if (PreviousClock > DummyClock)
            MinuteCounter++;
         PreviousClock = DummyClock;
         CommunicationToBasic.ServoClock = (DummyClock/1000 + 60 * MinuteCounter);

         choice = 1;

         inpAxes.I1_AXES.AXIS_NR = 1;
         (*b_axes) (choice, &status, &inpAxes, &outAxes);
         CommunicationToBasic.AxisPos[0] = outAxes.O1_AXES.ISPOS_MACHINE;

         inpAxes.I1_AXES.AXIS_NR = 2;
         (*b_axes) (choice, &status, &inpAxes, &outAxes);
         CommunicationToBasic.AxisPos[1] = outAxes.O1_AXES.ISPOS_MACHINE;

         inpAxes.I1_AXES.AXIS_NR = 3;
         (*b_axes) (choice, &status, &inpAxes, &outAxes);
         CommunicationToBasic.AxisPos[2] = outAxes.O1_AXES.ISPOS_MACHINE;

         inpAxes.I1_AXES.AXIS_NR = 4;
         (*b_axes) (choice, &status, &inpAxes, &outAxes);
         CommunicationToBasic.AxisPos[3] = outAxes.O1_AXES.ISPOS_MACHINE;

         inpAxes.I1_AXES.AXIS_NR = 5;
         (*b_axes) (choice, &status, &inpAxes, &outAxes);
         CommunicationToBasic.AxisPos[4] = outAxes.O1_AXES.ISPOS_MACHINE;
      }
      break;
```

```
        /*------------------Servo-----------------------------------------*/
      case E_TRANSFER:
      {
         CommunicationToBasic.NumberOfServoBlocksTransferred++;

      }
      break;

      /*------------------Servo-----------------------------------------*/

      default:
      break;
   }
}
```

Module wlprocon.c

The name of this module is abbreviation of "Welding Process Controller". The procedures in this module controls the welding process according to the inverse process model. The positions of the robot axes and the transformation from the robot co-ordinate system to the sensor co-ordinate system are used for determination of the position of the weld groove. The size of gap in the weld groove is calculated on the basis of measured positions of the workpieces in the weld groove, and welding control vectors are generated.

Source code:

```
/*****************************************************************************
File: WlProCon.c     Alias: Welding Process Controller.
*****************************************************************************/

#include "WlProCon.h"
#include "math.h"
#include "SCAN.h"
#include "Search.h"
#include "global.h"
#include "stdio.h"
#include "matlib.h"
#include "queue.h"


char *IpmFileName;
int RobotLocationFound;
IPM ipm;
int IpmType; /*IpmType is set to 0 for 'kalib.ipm' and to 1 for 'start.ipm'*/
struct ControlVector NewConVec;
extern RST_buffer rst_buffer;
RST_buffer  RstMachine;
char FilterStartPoint = 3;
char FilterInitiated = 3;
char Tack = FALSE;
extern int GPIBdummy;
int NewConVecCounter = 0;
int MeasErrorCounter = 0;
double SensorRotation_X;
double delta_Ly;
double L_lookahead;
double L_focus;
```

```
double L_sensor;
double PreviousB;
double PreviousC;
double PreviousX;
double PreviousY;
double PreviousZ;
double counter;
double Movement;
double Blockmove = 7.0 * 1000;
double DeltaCurve;
double CurveLength;
RST_buffer RstIndex[10];
char WaitDummy;
Plane Wp1Plane;
unsigned char SensorStatus;
unsigned char WelProConResponse;
char PosFile = FALSE;
char genposFile = TRUE;
char MonitorFile = TRUE;
char KinematFile = TRUE;
char TestFile = FALSE;
FILE *Test, *kinemat, *monitor, *pos, *genpos;
double MeasuredGap;
double SensorGap;
double FilterFactor;
double MaxPointToLineDeviation;
double WeldLength = 0;


TransformationMatrix sensor_T_machine, machine_T_sensor,
          reference_T_machine, machine_T_reference, machine_T_ic;

Point R1_machine, S1_machine, T1_machine, R2_machine, S2_machine,
       T2_machine, TCP_machine, Oic_start, Oic, Pfocus_ic, Pfocus_machine;

Vector R2S2, S2T2;

       /*********************/

void ResetWeldingProcessController(void)
{
       SensorStatus = StopVision();
       if (KinematFile)
       {
          fprintf(kinemat,"StopVision: %d\n", SensorStatus);
          fflush(kinemat);
       }
       SensorStatus = TurnLaserOff();
       if (KinematFile)
       {
          fprintf(kinemat,"TurnLaserOff: %d\n", SensorStatus);
          fflush(kinemat);
       }
       if (KinematFile)
       {
          fprintf(kinemat,"SensorStatus set to %d\n", SensorStatus);
          fflush(kinemat);
       }
       WelProConResponse = EndOfWCVQueue;
       SensorGap = 0;
       FilterInitiated = FilterStartPoint;
       WeldLength = 0;
```

```
        if (KinematFile)
        {
            fprintf(kinemat,"\n ResetWeldingProcessController is performed\n");
            fprintf(kinemat,"KINEMAT.LOG IS CLOSED");
            fclose(kinemat);
        }
        if (MonitorFile)
        {
            fprintf(monitor, "MONITOR.LOG IS CLOSED");
            fclose(monitor);
        }
        if (PosFile)
        {
            fprintf(pos, "POSITION.LOG IS CLOSED");
            fclose(pos);
        }
        if (genposFile)
        {
            fprintf(genpos, "genpos.LOG IS CLOSED");
            fclose(genpos);
        }

        if(TestFile)
        {
            fprintf(Test, "TEST.LOG IS CLOSED");
            fclose(Test);
        }
        ResetRobotLocationQueue();
        counter = 0;
}

/*************************************/

int CheckRstValidity(RST_buffer *sensor)
{
    double a1, b1, c1, a2, b2, c2, P1, P2, distance;

    makeLine(sensor->R1[1], sensor->R1[2], sensor->S1[1], sensor->S1[2], &a1, &b1, &c1);
    makeLine(sensor->R2[1], sensor->R2[2], sensor->S2[1], sensor->S2[2], &a2, &b2, &c2);
    distance = distBetweenPointAndLine(sensor->T1[1], sensor->T1[2], a1, b1, c1);
    if (counter < 3 || SensorStatus == 1 || distance < MaxPointToLineDeviation)
    {
        distance = distBetweenPointAndLine(sensor->T2[1], sensor->T2[2], a2, b2, c2);
        if (counter < 3 || SensorStatus == 1 || distance < MaxPointToLineDeviation)
        {
            distance = distBetweenPointAndLine(sensor->T1[1], sensor->T1[2], a2, b2, c2);
            if (counter < 3 || SensorStatus == 1 || distance < MaxPointToLineDeviation)
            {
                if (KinematFile)
                    fprintf(kinemat,"Distance between T1 an line (R1 to S1) %.2f:\n",
distance);
                    fprintf(kinemat,"Valid position of T1\n", SensorStatus);
                return TRUE;
            }
        }
    }
    if (KinematFile)
    {
        fprintf(kinemat,"Unproper position of T1 is corrected\n", SensorStatus);
        fflush(kinemat);
    }
    /*IntersectionBetweenLines(a1, b1, c1, a2, b2, c2, &P1, &P2);
    sensor->T1[1] = P1;
    sensor->T1[2] = P2;*/
    return FALSE;
}
```

```
/************************************/

double Filter(double Measurement)
{
   static double PrevReturnValue, ReturnValue;

   if (FilterInitiated == FilterStartPoint)
   {
      ReturnValue = Measurement;
      PrevReturnValue = ReturnValue;
      FilterInitiated = TRUE;
      if (KinematFile)
      {
         fprintf(kinemat,"FilterInitiated\n");
         fflush(kinemat);
      }
      return ReturnValue;
   }


   if (FilterInitiated == FALSE)
   {
      ReturnValue = PrevReturnValue;
      PrevReturnValue = ReturnValue;
      FilterInitiated = TRUE;
      if (KinematFile)
      {
         fprintf(kinemat,"FilterInitiated\n");
         fflush(kinemat);
      }
      return ReturnValue;
   }

   else
   {
      ReturnValue = (FilterFactor * PrevReturnValue) + ((1-FilterFactor) *
Measurement);
      PrevReturnValue = ReturnValue;
      return ReturnValue;
   }
}
      /*********************/

int DeltaY_Filter(double DeltaY_Value, double Distance)
{
   static double MeanDeltaY_Value;
   double FilterFactor = 0.6;
   if (CurveLength<180000)
   {
      MeanDeltaY_Value = DeltaY_Value/Distance;
      return 1;
   }
   else if (CurveLength < 190000)
   {
      MeanDeltaY_Value = FilterFactor * MeanDeltaY_Value + (1-FilterFactor) *
DeltaY_Value/Distance;
      return 1;
   }
   else if (sqrt(square((DeltaY_Value - MeanDeltaY_Value * Distance)/Distance)) < 0.02)
   {
      MeanDeltaY_Value = FilterFactor * MeanDeltaY_Value + (1-FilterFactor) *
DeltaY_Value/Distance;
      return 1;
   }
   else
      return 0;
}
```

```
/************************************/

void Calculate_sensor_T_machine(double X, double Y, double Z, double B, double C)
{
   TransformationMatrix Sensor_R_x, sensor_T_rest, sensor_T_reference;
   double Brad, Crad;
   Brad = (B/1000)*pi/180;
   Crad = (C/1000)*pi/180;
   if(KinematFile)
      fprintf(kinemat,"SensorRotation_X = %.1f\n",SensorRotation_X/pi*180);

   Sensor_R_x[0][0] = 1;
   Sensor_R_x[0][1] = 0;
   Sensor_R_x[0][2] = 0;
   Sensor_R_x[0][3] = 0;

   Sensor_R_x[1][0] = 0;
   Sensor_R_x[1][1] = cos(SensorRotation_X);
   Sensor_R_x[1][2] = -sin(SensorRotation_X);
   Sensor_R_x[1][3] = 0;

   Sensor_R_x[2][0] = 0;
   Sensor_R_x[2][1] = sin(SensorRotation_X);
   Sensor_R_x[2][2] = cos(SensorRotation_X);
   Sensor_R_x[2][3] = 0;
   /* sensor_R_x beskriver rotation af sensorframe omkring sensorens x-akse */

   sensor_T_rest[0][0] = 0;
   sensor_T_rest[0][1] = -1;
   sensor_T_rest[0][2] = 0;
   sensor_T_rest[0][3] = -L_lookahead;

   sensor_T_rest[1][0] = 1;
   sensor_T_rest[1][1] = 0;
   sensor_T_rest[1][2] = 0;
   sensor_T_rest[1][3] = -delta_Ly;

   sensor_T_rest[2][0] = 0;
   sensor_T_rest[2][1] = 0;
   sensor_T_rest[2][2] = 1;
   sensor_T_rest[2][3] = (L_focus - L_sensor);
   /* sensor_T_rest beskriver translation af det roterede sensorframe
      omkring sensorens x-akse til referenceframe + rotation 90grader om det
      nye sensorframe -> skal prikkes for at gore transformationen komplet*/

   matrixProd(Sensor_R_x, sensor_T_rest, &sensor_T_reference[0][0]);

   machine_T_reference[0][0] = cos(Brad) * cos(Crad);
   machine_T_reference[0][1] = -sin(Crad);
   machine_T_reference[0][2] = sin(Brad) * cos(Crad);
   machine_T_reference[0][3] = X;

   machine_T_reference[1][0] = cos(Brad) * sin(Crad);
   machine_T_reference[1][1] = cos(Crad);
   machine_T_reference[1][2] = sin(Brad) * sin(Crad);
   machine_T_reference[1][3] = Y;

   machine_T_reference[2][0] = -sin(Brad);
   machine_T_reference[2][1] = 0;
   machine_T_reference[2][2] = cos(Brad);
   machine_T_reference[2][3] = Z;

   Invers(machine_T_reference, &reference_T_machine[0][0]);
   matrixProd(sensor_T_reference, reference_T_machine, &sensor_T_machine[0][0]);
   Invers(sensor_T_machine, &machine_T_sensor[0][0]);
}
```

```
       /*********************/

void Calculate_RST_machine(void)
{
   int i;
   for(i=0;i<3;i++)
   {
      RstIndex[1].R1[i] = 1000 * RstIndex[1].R1[i];
      RstIndex[1].S1[i] = 1000 * RstIndex[1].S1[i];
      RstIndex[1].T1[i] = 1000 * RstIndex[1].T1[i];
      RstIndex[1].R2[i] = 1000 * RstIndex[1].R2[i];
      RstIndex[1].S2[i] = 1000 * RstIndex[1].S2[i];
      RstIndex[1].T2[i] = 1000 * RstIndex[1].T2[i];
   }

   pointTransformation(machine_T_sensor, RstIndex[1].R1, &RstMachine.R1[0]);
   pointTransformation(machine_T_sensor, RstIndex[1].S1, &RstMachine.S1[0]);
   pointTransformation(machine_T_sensor, RstIndex[1].T1, &RstMachine.T1[0]);
   pointTransformation(machine_T_sensor, RstIndex[1].R2, &RstMachine.R2[0]);
   pointTransformation(machine_T_sensor, RstIndex[1].S2, &RstMachine.S2[0]);
   pointTransformation(machine_T_sensor, RstIndex[1].T2, &RstMachine.T2[0]);
   /*RstMachine = FilterRstBuffer(RstMachine);*/
}

       /*********************/

void Calculate_TCP_in_machineCoordinates(void)
{
   Point TCP_reference;

   TCP_reference[0] = 0;
   TCP_reference[1] = 0;
   TCP_reference[2] = -L_focus;

   pointTransformation(machine_T_reference, TCP_reference, &TCP_machine[0]);
}

   /***********************************/

void DefineIcFrame(void)
{
   Vector dummyVec;
   static Vector i_ic, j_ic, k_ic;
   char dummyChar;

   Oic[0] = RstMachine.T1[0];
   Oic[1] = RstMachine.T1[1];
   Oic[2] = RstMachine.T1[2];

   makeVector(Oic_start,Oic,&i_ic[0]);
   unitVector(i_ic,&i_ic[0]);

   makeVector(Oic_start,RstMachine.R2,&dummyVec[0]);
   cross(dummyVec,i_ic,&k_ic[0]);
   unitVector(k_ic,&k_ic[0]);

   cross(k_ic,i_ic,&j_ic[0]);
   unitVector(j_ic,&j_ic[0]);

   for (dummyChar = 0; dummyChar < 3; dummyChar++)
   {
      machine_T_ic[dummyChar][0] = i_ic[dummyChar];
      machine_T_ic[dummyChar][1] = j_ic[dummyChar];
      machine_T_ic[dummyChar][2] = k_ic[dummyChar];
      machine_T_ic[dummyChar][3] = Oic[dummyChar];
   }
}
```

```
    /************************************/

int LowerInputRow(double MeasuredGap){
    int row = 0;
    if (KinematFile)
        fprintf(kinemat, "MeasuredGap for LowerInputRow: %f\n", MeasuredGap);
    while ((MeasuredGap > ipm.Input[row][0]) || (MeasuredGap == ipm.Input[row][0]))
    {
        if(row++ == ipm.NumOfRows)
        {
            if (KinematFile)
            {
                fprintf(kinemat,"ipm.NumOfRows = %d; Actual Row = %d\n", ipm.NumOfRows,
row);
                fprintf(kinemat,"fejl: Sensor m†ler gab, som er st›rre end modellens
max,\n");
                fprintf(kinemat,"eller ipm.NumOfRows for den anvendte procesmodel er
mindre end\n");
                fprintf(kinemat,"antallet af r'kker i den adaptive tabel.\n");
                fflush(kinemat);
            }
            return 666;
        }
    }
    return (row - 1);
}
    /************************************/

int UpperInputRow(double MeasuredGap){
    int row = (ipm.NumOfRows - 1);
    while ((MeasuredGap < ipm.Input[row][0]) || (MeasuredGap == ipm.Input[row][0]))
    {
        if(row-- < 0)
        {
            if (KinematFile)
            {
                fprintf(kinemat,"fejl: Sensor m†ler gab, som er mindre end 0,\n");
                fflush(kinemat);
            }
            return 666;
        }
    }
    return (row + 1);
}
    /************************************/
double Interpolate(int Coloumn, double MeasuredGap, int LowerRow, int UpperRow){
    double a;
    double b;
    double WCV_Value;


    /* Find the equation of the line which connects the WCV Values corre-
       sponding to the lower and larger gap: [WCV(gap) = a*gap + b] */
    a = (ipm.Wcv[UpperRow][Coloumn] - ipm.Wcv[LowerRow][Coloumn]);
    a = a/(ipm.Input[UpperRow][0] - ipm.Input[LowerRow][0]);

    b = ipm.Wcv[LowerRow][Coloumn] - a * ipm.Input[LowerRow][0];

    /* Find the interpolated WCV_Value which corresponds to the measured gap */

    WCV_Value = a * MeasuredGap + b;
    return WCV_Value;
}

    /************************************/
```

```
void DetermineWCV(double Gap, float *Wp1Dist, float *Wire, float *WelSpeed){
    int UpperRow;
    int LowerRow;
    double LaserPower;
    double IncidentBeamAngle;
    double DeltaDistance;
    double DeltaWire;
    double DeltaWelSpeed;

    LowerRow = LowerInputRow(Gap);
    UpperRow = UpperInputRow(Gap);
    if (Gap == ipm.Input[LowerRow][0])
    {
        /*LaserPower = ipm.Wcv[LowerRow][0];*/
        /*IncidentBeamAngle = ipm.Wcv[LowerRow][3];*/
        /*DefocusLength = ipm.Wcv[LowerRow][1]; */
        *Wire = ipm.Wcv[LowerRow][4];
        *WelSpeed = ipm.Wcv[LowerRow][5];
        *Wp1Dist = ipm.Wcv[LowerRow][2];
    }

    else if (Gap == ipm.Input[UpperRow][0])
    {
        /*LaserPower = ipm.Wcv[UpperRow][0];*/
        /*DefocusLength = ipm.Wcv[UpperRow][1];*/
        /*IncidentBeamAngle = ipm.Wcv[UpperRow][3];*/
        *Wire = ipm.Wcv[UpperRow][4];
        *WelSpeed = ipm.Wcv[UpperRow][5];
        *Wp1Dist = ipm.Wcv[UpperRow][2];
    }

    else
    {
        /*LaserPower = Interpolate(0, Gap, LowerRow, UpperRow);*/
        /*DefocusLength = Interpolate(1, Gap, LowerRow, UpperRow); */
        /*IncidentBeamAngle = Interpolate(3, Gap, LowerRow, UpperRow);*/
        *Wire = Interpolate(4, Gap, LowerRow, UpperRow);
        *WelSpeed = Interpolate(5, Gap, LowerRow, UpperRow);
        *Wp1Dist = Interpolate(2, Gap, LowerRow, UpperRow);
        if (KinematFile)
            fprintf(kinemat, "Interpolation Finished\n");
    }
    if(Tack == TRUE)
    {
        if (MonitorFile)
            fprintf(monitor,"Tackprocedures are used\n");
        UpperRow = UpperInputRow(2.0);
        /*LaserPower = ipm.Wcv[UpperRow][0];*/
        /*DefocusLength = ipm.Wcv[UpperRow][1];*/
        /*IncidentBeamAngle = ipm.Wcv[UpperRow][3];*/
        DeltaWire = ipm.Wcv[UpperRow][4];
        DeltaDistance = ipm.Wcv[UpperRow][2];
        DeltaWelSpeed = ipm.Wcv[UpperRow][5];
        *Wire = *Wire + DeltaWire;
        *WelSpeed = *WelSpeed + DeltaWelSpeed;
        *Wp1Dist = *Wp1Dist + DeltaDistance;
        Tack = FALSE;
    }

}

    /************************************/
```

```
void CalculateInitialConVec(double Xpos, double Ypos, double Zpos, double Bpos, double
Cpos)
{
   double DeltaX, DeltaY, DeltaZ, Dist;
   char dummy;

   if (GPIBdummy > 498)
      MeasErrorCounter++ ;

   Calculate_sensor_T_machine(Xpos,Ypos,Zpos,Bpos,Cpos);
   Calculate_RST_machine();

   fprintf(kinemat, "Calulate_RST_machine er udfoert\n");
   fflush(kinemat);

   for (dummy = 0; dummy < 3; dummy++)
   {
      Oic_start[dummy] = RstMachine.T1[dummy];
   }
   Calculate_TCP_in_machineCoordinates();

   DeltaX = (RstMachine.T2[0] - TCP_machine[0])/2 ;
   DeltaY = (RstMachine.T2[1] - TCP_machine[1])/2 ;
   DeltaZ = (RstMachine.T2[2] - TCP_machine[2])/2 ;
   Dist = sqrt(DeltaX*DeltaX + DeltaY*DeltaY + DeltaZ*DeltaZ);

   NewConVec.AxisPosition[5] =   300 ;
   NewConVec.AxisPosition[23] =  Xpos + DeltaX ;
   NewConVec.AxisPosition[24] =  Ypos + DeltaY ;
   NewConVec.AxisPosition[25] =  Zpos + DeltaZ ;

   if(genposFile)
   {
      fprintf(genpos,"x = %.1f; y = %.1f;  z = %.1f \n",
      NewConVec.AxisPosition[23]/1000, NewConVec.AxisPosition[24]/1000,
      NewConVec.AxisPosition[25]/1000);
   }

   if(MonitorFile)
   {
      monitor = fopen("monitor.log","w+");
      fprintf(monitor, "MONITOR.LOG IS OPENED");
      fprintf(monitor, "CurveLength => SensorGap => Speed => Wire => Dist\n");
   }
   if(PosFile)
   {
      pos = fopen("position.log","w+");
      fprintf(pos, "POSITION.LOG IS OPENED\n");
   }
   if(genposFile)
   {
      genpos = fopen("genpos.log","w+");
      fprintf(genpos, "genpos.LOG IS OPENED\n");
      fprintf(genpos, "Denne fil indeholder genererede vaerdier, som er sendt til
robotten\n");
   }

   if(TestFile)
   {
      Test = fopen("Test.log","w+");
      fprintf(Test, "TEST.LOG IS OPENED");
   }
   if (KinematFile)
      fprintf(kinemat, "CurveLength: %.0f => Initial Search Control Vector is
Generated\n",CurveLength);
}
```

```
      /*********************/

double CalculateSensorGap()
{
   double a, b, c, gap;
   double runding = 0.0;
   a = (RstIndex[1].R1[2] - RstIndex[1].T1[2]) / (RstIndex[1].R1[1] -
RstIndex[1].T1[1]);
   b = -1;
   c = RstIndex[1].T1[2] - (a * RstIndex[1].T1[1]);
   gap = a * RstIndex[1].T2[1] - RstIndex[1].T2[2] + c;
   if (gap > 0)
   {
      gap = gap / (sqrt(square(a) + square(b)));
      gap = gap / 1000;
      if (gap > runding)
         gap = gap - runding;
      else gap = 0;
      if(gap > 2)
         gap = 2;
   }
   else
   {
      gap = (-1 * gap) / (sqrt(square(a) + square(b)));
      gap = gap / 1000;
      if (gap > runding)
         gap = gap - runding;
      else gap = 0;
      if(gap > 2)
         gap = 2;
   }
   fprintf(kinemat,"CalculateSensorGap is performed:  %lf\n",gap);
   if(gap < 0.2)
      gap = gap;
   return gap;
}
      /*********************/

void CalculateNewSCV(double Xpos, double Ypos, double Zpos, double Bpos, double Cpos)
{
   double DeltaX, DeltaY, DeltaZ, Dist;
   float Wp1Distance, WireFeedRate, WeldingSpeed;

   if (GPIBdummy > 498)
      MeasErrorCounter++ ;

   Calculate_sensor_T_machine(Xpos,Ypos,Zpos,Bpos,Cpos);
   Calculate_RST_machine();
   DefineIcFrame();
   makePlane(Oic_start, RstMachine.T1, RstMachine.R1, &Wp1Plane[0]);
   MeasuredGap = distBetweenPointAndPlane(Wp1Plane, RstMachine.T2)/1000;
   SensorGap = CalculateSensorGap();
   SensorGap = Filter(SensorGap);
   if (KinematFile)
      fprintf(kinemat,"CurveLength: %.0f:  MeasuredGap: %.1f;    SensorGap: %.1f\n",
CurveLength/1000, MeasuredGap/ 1000, SensorGap);
   Calculate_TCP_in_machineCoordinates();

   DetermineWCV(SensorGap, &Wp1Distance, &WireFeedRate, &WeldingSpeed);
   WeldingSpeed = 200;
   /*if (KinematFile)
   {
      fprintf(kinemat,"SensorGap = %f =>\n",SensorGap);
      fprintf(kinemat,"Wp1Distance = %f\n", Wp1Distance);
      fprintf(kinemat,"WireFeedRate = %f\n",WireFeedRate);
      fprintf(kinemat,"WeldingSpeed= %f\n", WeldingSpeed);
   } */
```

```
    Pfocus_ic[0] = 0;
    Pfocus_ic[1] = -1000 * (SensorGap + Wp1Distance);
    Pfocus_ic[2] = 0;
    pointTransformation(machine_T_ic, Pfocus_ic, &Pfocus_machine[0]);

    DeltaX = Pfocus_machine[0] - TCP_machine[0] ;
    DeltaY = Pfocus_machine[1] - TCP_machine[1] ;
    DeltaZ = Pfocus_machine[2] - TCP_machine[2] ;
    Dist = sqrt(DeltaX*DeltaX + DeltaY*DeltaY + DeltaZ*DeltaZ);
    /*if ((Movement > 10) && TestFile)
       fprintf(Test, "CurveLength: %.0f =>    DeltaX: %f; DeltaY: %f; DeltaZ: %f\n",
          CurveLength, DeltaX/Dist, DeltaY/Dist, DeltaZ/Dist);*/

    NewConVec.AxisPosition[5] =   300 ;
    NewConVec.AxisPosition[23] =  Xpos + DeltaX ;
    NewConVec.AxisPosition[24] =  Ypos + DeltaY ;
    NewConVec.AxisPosition[25] =  Zpos + DeltaZ ;

    if(genposFile)
    {
       fprintf(genpos,"x = %.1f; y = %.1f;  z = %.1f \n",
       NewConVec.AxisPosition[23]/1000, NewConVec.AxisPosition[24]/1000,
       NewConVec.AxisPosition[25]/1000);
    }
    /*if (KinematFile)
    {
       fprintf(kinemat,"F = %f\n", NewConVec.AxisPosition[5]);
       fprintf(kinemat,"Xpos = %f\n", NewConVec.AxisPosition[23]);
       fprintf(kinemat,"Ypos = %f\n", NewConVec.AxisPosition[24]);
       fprintf(kinemat,"Zpos = %f\n", NewConVec.AxisPosition[25]);
    } */
    if(MonitorFile)
    {
       fprintf(monitor, "CurveLength: %.0f => Gap = %.2f;   Speed: %.0f;   Wire: %.0f;
Wp1Dist: %.2f\n",
          CurveLength/1000, SensorGap, WeldingSpeed, WireFeedRate, Wp1Distance);
    }

    if(PosFile)
    {
       fprintf(pos, "CL: %.1f T1[1]: %.2f;  T1[2]: %.2f;  T1[3]: %.2f\n ",
       CurveLength/1000, RstMachine.T1[0]/1000, RstMachine.T1[1]/1000,
RstMachine.T1[2]/1000);
    }
}
       /*********************/
void CalculateNewWCV(double Xpos, double Ypos, double Zpos, double Bpos, double Cpos)
{
    double DeltaX, DeltaY, DeltaZ, Dist;
    float Wp1Distance, WireFeedRate, WeldingSpeed;

    if (GPIBdummy > 498)
       MeasErrorCounter++ ;

    Calculate_sensor_T_machine(Xpos,Ypos,Zpos,Bpos,Cpos);
    Calculate_RST_machine();
    DefineIcFrame();

    makePlane(Oic_start, RstMachine.T1, RstMachine.R1, &Wp1Plane[0]);
    MeasuredGap = distBetweenPointAndPlane(Wp1Plane, RstMachine.T2)/1000;

    SensorGap = CalculateSensorGap();
    if (MonitorFile)
    {
     /*  fprintf(monitor, "CurveLength: %.0f => RawGap = %.2f;\n",
       CurveLength/1000, SensorGap);                                */
    }
```

```
   SensorGap = Filter(SensorGap);
   if (KinematFile)
      fprintf(kinemat,"CurveLength: %.0f:  MeasuredGap: %.1f;    SensorGap: %.1f\n",
CurveLength/1000, MeasuredGap/ 1000, SensorGap);
   DetermineWCV(SensorGap, &Wp1Distance, &WireFeedRate, &WeldingSpeed);


   Pfocus_ic[0] = 0;
   Pfocus_ic[1] = -1000 * (SensorGap + Wp1Distance);
   Pfocus_ic[2] = 0;
   pointTransformation(machine_T_ic, Pfocus_ic, &Pfocus_machine[0]);

   Calculate_TCP_in_machineCoordinates();

   DeltaX = Pfocus_machine[0] - TCP_machine[0] ;
   DeltaY = Pfocus_machine[1] - TCP_machine[1] ;
   DeltaZ = Pfocus_machine[2] - TCP_machine[2] ;

   Dist = sqrt(DeltaX*DeltaX + DeltaY*DeltaY + DeltaZ*DeltaZ);

   /*if(TestFile)
      fprintf(Test, "CurveLength: %.0f =>    DeltaX: %f; DeltaY: %f; DeltaZ: %f\n",
         CurveLength, DeltaX/Dist, DeltaY/Dist, DeltaZ/Dist);*/

   if(DeltaY_Filter(DeltaY, Dist) == 1)
   {
      NewConVec.AxisPosition[5] =   WeldingSpeed ;
      NewConVec.AxisPosition[18] =  WireFeedRate ;
      NewConVec.AxisPosition[23] =  Xpos + DeltaX ;
      NewConVec.AxisPosition[24] =  Ypos + DeltaY ;
      NewConVec.AxisPosition[25] =  Zpos + DeltaZ ;

      if(genposFile)
      {
         fprintf(genpos,"x = %.1f; y = %.1f;  z = %.1f \n",
         NewConVec.AxisPosition[23]/1000, NewConVec.AxisPosition[24]/1000,
         NewConVec.AxisPosition[25]/1000);
      }
   }
   else
      if(TestFile)
      {
         fprintf(Test,"CurveLength: %f:   CONTROLVECTOR IS NOT EXECUTED BECAUSE OF BIG
DELTAY\n",
         CurveLength/1000);
      }

   if (MonitorFile)
   {
      fprintf(monitor, "CurveLength: %.0f => Gap = %.2f;   Speed: %.0f;   Wire: %.0f;
Wp1Dist: %.2f\n",
      CurveLength/1000, SensorGap, WeldingSpeed, WireFeedRate, Wp1Distance);
   }

   if(PosFile)
   {
      fprintf(pos, "CL: %.1f T1[1]: %.2f;  T1[2]: %.2f;  T1[3]: %.2f\n ",
      CurveLength/1000, RstMachine.T1[0]/1000, RstMachine.T1[1]/1000,
RstMachine.T1[2]/1000);
   }
}

      /*********************/
```

```
double CalculateDeltaCurve(double B, double C, double X,double Y,double Z)
{
   DeltaCurve = square(X-PreviousX) + square(Y-PreviousY) + square(Z-PreviousZ);
   DeltaCurve = sqrt(DeltaCurve);
   CurveLength = CurveLength + DeltaCurve;
   PreviousX = X;
   PreviousY = Y;
   PreviousZ = Z;
   return(DeltaCurve);
   /* Denne funktion skal udbygges til at indeholde B og C vinklerne */
}
   /************************************/

void InitializeScannerPosition(double C_angle)
{
   FILE *scanpos;
   int j;
   double Parameter;

   if ((scanpos = fopen("scanpos.INI", "rt"))
       == NULL)
   {
      fprintf(stderr, "Cannot open scanpos");
   }
      /* read the scanner position data */

   for(j = 0; j < 16; j++) /* Get to SensorRotation_X */
   {
      fscanf(scanpos, "%*s");
   }
   fscanf(scanpos, "%lf", &Parameter);
   SensorRotation_X = Parameter/180 * pi;

   for(j = 0; j < 8; j++) /* Get to delta_Ly */
   {
      fscanf(scanpos, "%*s");
   }
   fscanf(scanpos, "%lf", &Parameter);
   delta_Ly = Parameter * 1000;


   for(j = 0; j < 2; j++) /* Get to L_sensor */
   {
      fscanf(scanpos, "%*s");
   }
   fscanf(scanpos, "%lf", &Parameter);
   L_sensor = (Parameter - 30) * 1000;

   for(j = 0; j < 2; j++) /* Get to L_lookahead */
   {
      fscanf(scanpos, "%*s");
   }
   fscanf(scanpos, "%lf", &Parameter);
   L_lookahead = Parameter * 1000;

   for(j = 0; j < 7; j++) /* Get to L_focus */
   {
      fscanf(scanpos, "%*s");
   }
   fscanf(scanpos, "%lf", &Parameter);
   L_focus = Parameter * 1000;

   for(j = 0; j < 9; j++) /* Get to L_focus */
   {
      fscanf(scanpos, "%*s");
   }
   fscanf(scanpos, "%lf", &Parameter);
```

```
   MaxPointToLineDeviation = Parameter;

   fscanf(scanpos, "%*s"); /* Get to MaxPointToLineDeviation */
   fscanf(scanpos, "%lf", &Parameter);
   FilterFactor = Parameter;

   fclose(scanpos);
}


   /***********************************/

void chooseIpm(int IpmNumber)
{
   if (IpmNumber == 0)
      IpmFileName = "start.ipm";

   if (IpmNumber == 1)
      IpmFileName = "kalib.ipm";

   if(KinematFile)
   {
      fprintf(kinemat, "chooseIpm performed\n");
      fprintf(kinemat, "Ipmnumber: %d; IpmFileName: %s\n", IpmNumber, IpmFileName);
      fflush(kinemat);
   }
}

/*******************************************************/

int InstantiateIpm(void)
{
   FILE *InverseProMod;
   int i,j;
   double Parameter;
   int InstantiateStatus = FALSE;

   if(KinematFile)
   {
      fprintf(kinemat, "InstantiatingIpm\n");
      fprintf(kinemat, "IpmFileName: %s\n", IpmFileName);
      fflush(kinemat);
   }

   ipm.NumOfRows = 6;

   for (i = 0; i < 20; i++)
   {
      if ((InverseProMod = fopen(IpmFileName, "rt")) == NULL)
      {
         InstantiateStatus = FALSE;
         if (KinematFile)
         {
            fprintf(kinemat,"Reading IPM failed: %d times\n", i);
            fflush(kinemat);
         }
         fclose(InverseProMod);
      }
      else
      {
         InstantiateStatus = TRUE;
         if (KinematFile)
            fprintf(kinemat,"Reading IPM Succeded\n");
         i = 20;
      }
   }
   if (InstantiateStatus == FALSE)
```

```
    {
        if(KinematFile)
        {
            fprintf(kinemat, "Cannot open IPM\n");
            fflush(kinemat);
        }
        return FALSE;
    }
        /* read the data and display it */
    else
    {
        for(j = 0; j < 31; j++) /* Get to the adaptive table */
        {
            fscanf(InverseProMod, "%*s");
        }

        for(i = 0; i < 6; i++) /* Read the rows of adaptive table */
        {
            fscanf(InverseProMod, "%lf", &Parameter);
            ipm.Input[i][0] = Parameter;
            fscanf(InverseProMod, "%lf", &Parameter);
            ipm.Wcv[i][1] = Parameter;
            fscanf(InverseProMod, "%lf", &Parameter);
            ipm.Wcv[i][2] = Parameter;
            fscanf(InverseProMod, "%lf", &Parameter);
            ipm.Wcv[i][4] = Parameter;
            fscanf(InverseProMod, "%lf", &Parameter);
            ipm.Wcv[i][5] = Parameter;
        }
        fscanf(InverseProMod, "%*s");
        fscanf(InverseProMod, "%lf", &Parameter);
        WeldLength = Parameter;
        fclose(InverseProMod);
        if(KinematFile)
        {
            fprintf(kinemat, "IntantiateIpm completed succesfully\n");
            fflush(kinemat);
        }
        return TRUE;
    }
}

    /************************************/

int SetSensorTemplate(double C_angle)
{
    unsigned char dummy[] =  "fillet";
    InitializeScannerPosition(C_angle);
    if (KinematFile)
    {
        kinemat = fopen("kinemat.log","w+");
        fprintf(kinemat,"KINEMATFILE IS OPENED\n");
        fflush(kinemat);
    }
    if(InstantiateIpm() == FALSE)
        ResetWeldingProcessController();
    if (KinematFile)
    {
        fprintf(kinemat,"before 'set template'\n");
        fflush(kinemat);
    }

    SensorStatus = SetTemplate(dummy);
    if (SensorStatus == 0)
    {
```

```
        if (KinematFile)
        {
            fprintf(kinemat,"SetTemplate %s: %d\n", dummy, SensorStatus);
            fflush(kinemat);
        }
        WelProConResponse = SensorTemplateSet;
    }
    else
    {
        if (KinematFile)
        {
            fprintf(kinemat,"SetTemplate %s failed: %d\n", dummy, SensorStatus);
            fflush(kinemat);
        }
        WelProConResponse = WaitingForSensor;
    }
}

        /*********************/

int SyncronizeSensorTime(void)
{
    if (KinematFile)
    {
        fprintf(kinemat,"before 'syncronizeTime'\n");
        fflush(kinemat);
    }

    SensorStatus = SyncronizeTime();
    if (SensorStatus == 0)
    {
     WelProConResponse = SensorTimeSyncronized;
    }
    else
    {
        if (KinematFile)
        {
            fprintf(kinemat,"SyncronizeTime failed: %d\n", SensorStatus);
            fflush(kinemat);
        }
        WelProConResponse = WaitingForSensor;
    }
}
        /*********************/
int InitializeSensor(void)
{
    char dummy, index;
    unsigned char LostSensorAcknowledge;
    if (KinematFile)
    {
        fprintf(kinemat,"before 'TurnLaserOn'\n");
        fflush(kinemat);
    }

    SensorStatus = TurnLaserOn();
    if (KinematFile)
    {
        fprintf(kinemat,"TurnLaserOn: %d\n", SensorStatus);
        fflush(kinemat);
    }
    LostSensorAcknowledge = SensorStatus;
     /*****/
    if (KinematFile)
    {
        fprintf(kinemat,"before 'StartVision'\n");
        fflush(kinemat);
    }
```

```
    SensorStatus = StartVision();
    if (KinematFile)
    {
        fprintf(kinemat,"StartVision: %d\n", SensorStatus);
        fflush(kinemat);
    }
    if (LostSensorAcknowledge == 0)
        LostSensorAcknowledge = SensorStatus;
     /*****/
    for(dummy = 0; dummy < 5; dummy++)
    {
        /*To Empty MVS buffer for old measurements*/
        SensorStatus = ReadRSTPoint();
        if (SensorStatus == 0)
            dummy = 5;
    }
    if (KinematFile)
    {
        fprintf(kinemat,"ReadInitialRST to empty MVS buffer: %d\n", SensorStatus);
        fflush(kinemat);
    }
    if (LostSensorAcknowledge == 0)
        LostSensorAcknowledge = SensorStatus;
     /*****/
    if (LostSensorAcknowledge == 0)
    {
        CurveLength = 0;
        WelProConResponse = SensorInitialized;
        if (KinematFile)
            fprintf(kinemat,"SensorInitialized\n");
    }
    else
    {
        if (KinematFile)
            fprintf(kinemat,"InitializeSensor Failed\n");
        SensorStatus = StopVision();
        if (KinematFile)
            fprintf(kinemat,"StopVision: %d\n", SensorStatus);
        SensorStatus = TurnLaserOff();
        if (KinematFile)
        {
            fprintf(kinemat,"TurnLaserOff: %d\n", SensorStatus);
            fflush(kinemat);
        }
        WelProConResponse = WaitingForSensor;
    }
}

        /*********************/

    /*Preparing Measurement For Initial SCV, by filling RstIndex*/

int InitializeRstIndex(void)
{
    static char index = 0;
    char dummy;
    for(dummy = 0; dummy < 5; dummy++)
    {
        SensorStatus = ReadRSTPoint();
        if (SensorStatus == 0)
            dummy = 5;
    }
    if (SensorStatus == 0)
    {
        RstIndex[index] = rst_buffer;
        index++;
```

```
        if (index > 9)
        {
            index = 0;
            WelProConResponse = InitializingRstIndexFinished;
        }
        else
            WelProConResponse = InitializingRstIndex;
    }
    else
        WelProConResponse = WaitingForSensor;
}

        /*********************/

int GetInitSCV(void)
{
    double Xpos, Ypos, Zpos, Bpos, Cpos;
    char dummy;
    if (KinematFile)
    {
        fprintf(kinemat,"GetInitSCV at CurveLength: %f\n", CurveLength);
        fflush(kinemat);
    }
    for(dummy = 0; dummy < 5; dummy++)
    {
        SensorStatus = ReadRSTPoint();
        if (KinematFile)
            fprintf(kinemat,"SensorStatus = %d\n",SensorStatus);
        if ((SensorStatus == 0) && (CheckRstValidity(&rst_buffer) == TRUE))
            dummy = 5;
    }
    for (dummy = 0; dummy < 10; dummy++)
    {
        RstIndex[dummy] = RstIndex[dummy + 1];
    }
    RstIndex[9] = rst_buffer;
    if((RstIndex[8].t > RstIndex[9].t) || (RstIndex[7].t > RstIndex[8].t))
        RstIndex[8] = RstIndex[9];
    if (SensorStatus == 0)
    {
        RobotLocationFound = tryFindRobotPosition(RstIndex[1].t, &Xpos, &Ypos, &Zpos,
&Bpos, &Cpos);
        if (RobotLocationFound == TRUE)
        {
            PreviousB = Bpos;
            PreviousC = Cpos;
            PreviousX = Xpos;
            PreviousY = Ypos;
            PreviousZ = Zpos;

            CalculateInitialConVec(Xpos, Ypos, Zpos, Bpos, Cpos);
            fprintf(kinemat, "CalculateInitialConVec er udfoert\n");
            fflush(kinemat);

            WelProConResponse = InitSCVGenerated;
            counter = 0;
        }
        else
            WelProConResponse = WaitingForRobot;
    }
    else
        WelProConResponse = WaitingForSensor;
}

        /*********************/

int GetSuccSCV(double Xpos, double Ypos, double Zpos, double Bpos, double Cpos)
```

```
{
    char dummy;
    if (KinematFile)
        fprintf(kinemat,"GetSCV\n");
    Movement = Movement + CalculateDeltaCurve(Bpos, Cpos, Xpos, Ypos, Zpos);
    if ((counter < 5) || (Movement > Blockmove))
    {
        if (KinematFile)
            fprintf(kinemat,"Movement = %f\n",Movement/1000);
        SensorStatus = ReadRSTPoint();
        if ((SensorStatus == 0) && (CheckRstValidity(&rst_buffer) == TRUE))
        {
            for (dummy = 0; dummy < 9; dummy++)
                RstIndex[dummy] = RstIndex[dummy + 1];
            RstIndex[9] = rst_buffer;
            if((RstIndex[8].t > RstIndex[9].t) || (RstIndex[7].t > RstIndex[8].t))
                RstIndex[8] = RstIndex[9];

            RobotLocationFound = tryFindRobotPosition(RstIndex[1].t, &Xpos, &Ypos, &Zpos,
&Bpos, &Cpos);
            if(RobotLocationFound == TRUE)
            {
                if (KinematFile)
                {
                    fprintf(kinemat,"RobotLocation is Found for actual Sensor
Measurement\n");
                    fflush(kinemat);
                }
                CalculateNewWCV(Xpos, Ypos, Zpos, Bpos, Cpos);
                /*CalculateNewSCV(Xpos, Ypos, Zpos, Bpos, Cpos);*/
                WelProConResponse = SuccSCVGenerated;
                if (KinematFile)
                    fprintf(kinemat, "ScvGenerated\n");
                Movement = 0;
                counter++;
            }
        }
        else
        {
            WelProConResponse = WaitingForSensor;
            if (KinematFile)
                fprintf(kinemat, "WaitingForSensor in GetSuccSCV\n");
        }
    }
    else
    {
        WelProConResponse = WaitingForRobot;
        if (KinematFile)
            fprintf(kinemat, "WaitingForRobot in GetSuccWcv\n");
    }
    if (counter == 5)
        WelProConResponse = StartPointDefined;
}
        /*********************/

int GetWCV(double Xpos, double Ypos, double Zpos, double Bpos, double Cpos)
{
    char dummy, meascounter = 0;
    if (KinematFile)
        /*fprintf(kinemat,"GetWCV\n");*/
    Movement = Movement + CalculateDeltaCurve(Bpos, Cpos, Xpos, Ypos, Zpos);
    if (Movement > Blockmove)
    {
        if (KinematFile)
            fprintf(kinemat,"Movement = %f\n",Movement/1000);
        SensorStatus = ReadRSTPoint();
```

```
   if (SensorStatus == 1)
      {
         Tack = TRUE;
         FilterInitiated = FALSE;
         if (MonitorFile)
            fprintf(monitor,"Filter is initiated\n");
         while(meascounter < 12)
         {
            SensorStatus = ReadRSTPoint();
            if ((SensorStatus == 0) && (CheckRstValidity(&rst_buffer) == TRUE))
            {
               for (dummy = 0; dummy < 9; dummy++)
               RstIndex[dummy] = RstIndex[dummy + 1];
               RstIndex[9] = rst_buffer;
               meascounter++;
            }
            if((RstIndex[8].t > RstIndex[9].t) || (RstIndex[7].t > RstIndex[8].t))
               RstIndex[8] = RstIndex[9];
         }
      }
      else if ((SensorStatus == 0) && (CheckRstValidity(&rst_buffer) == TRUE))
      {
         for (dummy = 0; dummy < 9; dummy++)
            RstIndex[dummy] = RstIndex[dummy + 1];
         RstIndex[9] = rst_buffer;
         if((RstIndex[8].t > RstIndex[9].t) || (RstIndex[7].t > RstIndex[8].t))
            RstIndex[8] = RstIndex[9];


         RobotLocationFound = tryFindRobotPosition(RstIndex[1].t, &Xpos, &Ypos, &Zpos,
&Bpos, &Cpos);
         if(RobotLocationFound == TRUE)
         {
            if (KinematFile)
               fprintf(kinemat,"RobotLocation is Found for actual Sensor
Measurement\n");
            CalculateNewWCV(Xpos, Ypos, Zpos, Bpos, Cpos);
            WelProConResponse = WCVGenerated;
            if (KinematFile)
               fprintf(kinemat, "WcvGenerated\n");
            Movement = 0;
            counter++;
         }
      }
      else
      {
         WelProConResponse = WaitingForSensor;
         if (KinematFile)
         {
            fprintf(kinemat, "WaitingForSensor: %d in GetWcv\n", SensorStatus);
            fprintf(kinemat, "rst_buffer.t: %.2f; RstIndex[9].t: %.2f\n",
rst_buffer.t, RstIndex[9].t);
         }
      }
   }
   else
   {
      WelProConResponse = WaitingForRobot;
      /*if (KinematFile)
         fprintf(kinemat, "WaitingForRobot in GetWcv\n");*/
   }
   if (CurveLength >= WeldLength * 1000 )
      ResetWeldingProcessController();
}
```

Module wlprocon.h

This module is a header file for wlprocon.c

Source code

```
/*****************************************************************************

File: WlProCon.h     Alias: Header file for Welding Process Controller.

*****************************************************************************/
/*****************************************************************************
  Definition of variables:
*****************************************************************************/


 #define InitSCVGenerated 50
 #define SuccSCVGenerated 51
 #define WCVGenerated   52
 #define StartPointDefined 53
 #define WaitingForRobot 54
 #define EndOfWCVQueue  55
 #define WaitingForSensor 56
 #define SensorInitialized 57
 #define SensorTimeSyncronized 58
 #define SensorTemplateSet 59
 #define KalibrationInitialized 60
 #define KalibrationOngoing 61
 #define KalibrationFinished 62
 #define InitializingRstIndex 63
 #define InitializingRstIndexFinished 64
/*****************************************************************************
  Definition of functions:
*****************************************************************************/

void ResetWeldingProcessController(void);

void chooseIpm(int IpmNumber);

int SetSensorTemplate(double C_angle);

int SyncronizeSensorTime(void);

int InitializeSensor(void);

int InitializeRstIndex(void);

int GetInitSCV(void);

int GetSuccSCV(double X, double Y, double Z, double B, double C);

int GetWCV(double X, double Y, double Z, double B, double C);

void UpdateRobotLocations(double T, double X, double Y, double Z, double B, double C);

int FindRobotPosition(double T, double *LocalXPos, double *LocalYPos,
      double *LocalZPos, double *LocalBPos, double *LocalCPos);
```

Module Scan.c

This software is a high level interface for communication with an external hardware platform through a GPIB interface. The laser range sensor is

controlled through an external hardware platform and communication between the user written procedures for laser welding control and the software for the laser range sensor is therefore implemented by using a GPIB interface (General Purpose Interface Bus). The GPIB interface consists of a hardware interface board on each platform, connected by cables, and software for controlling this board on each platform.

Source Code

```
/*********************************************
Filename: SCAN.c
*********************************************/
#include "dos.h"
#include "stdio.h"
#include "scan.h"
#include "GPIBdrv.h"
#include "global.h"

extern RD rd;
WR wr;
extern int ibcnt;
RSTpoints rstpoint[6];
Transfer_buffer transfer_buffer;
RST_buffer rst_buffer;
int loop;
char MvsFile = TRUE;
FILE *mvs;

/******************/
void DelayFunction(int DelayCounts)
{
   int i;
   for(i = 0; i < DelayCounts; i++);
}


/******************/
unsigned char TurnLaserOn(void)
{
   ibonl(1);         /*Initialize GPIB board*/
   DelayFunction(10000);
   ibsic();          /*Set interface clear (local GPIB board)*/
   DelayFunction(10000);
   ibsre(1);         /*Set remote enable*/
   DelayFunction(10000);
   ibcmd("?_@(",4);  /*Set GPIB board to talk and sensor GPIB board to listen*/
   DelayFunction(10000);
   wr[0]=6;          /*Code for Turn on laser*/
   wr[1]=0;          /*Stop bit*/
   ibwrt(wr,((long)strlen(&wr[1])+1));
   DelayFunction(10000);
   ibcmd("?_H ",4);  /*Set GPIB board to listen and sensor GPIB board to talk*/
   DelayFunction(10000);
   ibrd(rd,2000L);   /*Read response from scanner*/
   DelayFunction(10000);
   if(MvsFile)
   {
      fprintf(mvs, "Turn Laser On: %d\n", rd[0]);
   }
   return rd[0];
}
```

```
/******************/

unsigned char TurnLaserOff(void)
{
   ibcmd("?_@(",4);  /*Set GPIB board to talk and sensor GPIB board to listen*/
   DelayFunction(10000);
   wr[0]=7;          /*Code for Turn on laser*/
   wr[1]=0;          /*Stop bit*/
   ibwrt(wr,((long)strlen(&wr[1])+1));
   DelayFunction(10000);
   ibcmd("?_H ",4);  /*Set GPIB board to listen and sensor GPIB board to talk*/
   DelayFunction(10000);
   ibrd(rd,2000L);   /*Read response from scanner*/
   DelayFunction(10000);
   if(MvsFile)
   {
      fprintf(mvs, "Turn Laser Off: %d\n", rd[0]);
      fprintf(mvs, "MVS.LOG IS CLOSED");
      fclose(mvs);
   }
   return rd[0];
}



/******************/

unsigned char StartVision(void)
{
   ibcmd("?_@(",4);  /*Set GPIB board to talk and sensor GPIB board to listen*/
   DelayFunction(10000);
   wr[0]=1;          /*Code for Start sensor*/
   wr[1]=0;          /*Stop bit*/
   ibwrt(wr,((long)strlen(&wr[1])+1));
   DelayFunction(10000);
   ibcmd("?_H ",4);  /*Set GPIB board to listen and sensor GPIB board to talk*/
   ibrd(rd,2000L);   /*Read response from scanner*/
   DelayFunction(10000);
   if(MvsFile)
   {
      fprintf(mvs, "Start Vision: %d\n", rd[0]);
   }
   return rd[0];
}


/******************/

unsigned char StopVision(void)
{
   ibcmd("?_@(",4);  /*Set GPIB board to talk and sensor GPIB board to listen*/
   DelayFunction(10000);
   wr[0]=0;          /*Code for Start sensor*/
   wr[1]=0;          /*Stop bit*/
   ibwrt(wr,((long)strlen(&wr[1])+1));
   DelayFunction(10000);
   ibcmd("?_H ",4);  /*Set GPIB board to listen and sensor GPIB board to talk*/
   ibrd(rd,2000L);   /*Read response from scanner*/
   DelayFunction(10000);
   if(MvsFile)
   {
      fprintf(mvs, "Stop Vision: %d\n", rd[0]);
   }
   return rd[0];
}
```

```
/******************/

unsigned char SyncronizeTime(void)
{
   ibcmd("?_@(",4);  /*Set GPIB board to talk and sensor GPIB board to listen*/
   DelayFunction(2000);
   wr[0]=9;          /*Code for Syncronization of Time Stamp*/
   wr[1]=0;          /*Stop bit*/
   ibwrt(wr,((long)strlen(&wr[1])+1));
   DelayFunction(2000);
   ibcmd("?_H ",4);  /*Set GPIB board to listen and sensor GPIB board to talk*/
   DelayFunction(2000);
   ibrd(rd,2000L);   /*Read response from scanner*/
   DelayFunction(2000);
 /*  if(MvsFile)
   {
   fprintf(mvs, "Syncronize Time: %d\n", rd[0]); remmet ud for at spare tid
   } */
   return rd[0];
}

/******************/

unsigned char SetTemplate(unsigned char TemplateName[81])
{
   wr[0]=2;          /*Code for Turn on laser*/
   strcat(wr, TemplateName);

   ibonl(1);         /*Initialize GPIB board*/
   DelayFunction(10000);
   ibsic();          /*Set interface clear (local GPIB board)*/
   DelayFunction(10000);
   ibsre(1);         /*Set remote enable*/
   DelayFunction(10000);
   ibcmd("?_@(",4);  /*Set GPIB board to talk and sensor GPIB board to listen*/
   DelayFunction(10000);
   ibwrt(wr,((long)strlen(&wr[1])+1));
   DelayFunction(10000);
   ibcmd("?_H ",4);  /*Set GPIB board to listen and sensor GPIB board to talk*/
   DelayFunction(10000);
   ibrd(rd,2000L);   /*Read response from scanner*/
   DelayFunction(10000);
   if(MvsFile)
   {
      mvs = fopen("mvs.log","w+");
      fprintf(mvs, "MVS.LOG IS OPENED");
      fprintf(mvs, "Set Template (%s): %d\n", TemplateName, rd[0]);
   }
   return rd[0];
}

/******************/

unsigned char ReadRSTPoint(void)
{
   unsigned int i;
   wr[0]=3;          /*Code for Reading RSTpoints from the Sensor*/
   wr[1]=0;
   ibcmd("?_@(",4);  /*Set GPIB board to talk and sensor GPIB board to listen*/
   DelayFunction(5000);
   ibwrt(wr,(long)(strlen(&wr[1])+1));
   DelayFunction(5000);
   ibcmd("?_H ",4);  /*Set GPIB board to listen and sensor GPIB board to talk*/
   DelayFunction(5000);
   ibrd(rd,2000L);   /*Read response from scanner*/
   DelayFunction(5000);
```

```
    if(MvsFile)
    {
       fprintf(mvs, "Read RST Point: %d\n", rd[0]);
    }
    ExtractRSTPoints();
    return rst_buffer.status;
}

void ExtractRSTPoints()
{
    char dummy;
    transfer_buffer.status = (rd[0]);

    for(loop = 0; loop <= 5; loop++)
    {
       rstpoint[loop].u = *(float *)(&rd[1 + (loop*8)]);  /*0 8 16 24 32 40 48 */
       rstpoint[loop].v = *(float *)(&rd[5 + (loop*8)]);  /*4 12 20 28 36 44 52*/
    }
    transfer_buffer.Time = *(float *)(&rd[ibcnt-4]);
    /*  Move the RSTpoint structure into the Transferbuffer structure. */
    transfer_buffer.R1 = rstpoint[0];
    transfer_buffer.S1 = rstpoint[1];
    transfer_buffer.T1 = rstpoint[2];
    transfer_buffer.T2 = rstpoint[5];
    transfer_buffer.S2 = rstpoint[4];
    transfer_buffer.R2 = rstpoint[3];
    /*
     * Move Transferbuffer into Replybuffer
     */
    rst_buffer.R1[0] = 0;
    rst_buffer.R1[1] = rstpoint[0].u;
    rst_buffer.R1[2] = rstpoint[0].v;
    rst_buffer.S1[0] = 0;
    rst_buffer.S1[1] = rstpoint[1].u;
    rst_buffer.S1[2] = rstpoint[1].v;
    rst_buffer.T1[0] = 0;
    rst_buffer.T1[1] = rstpoint[2].u;
    rst_buffer.T1[2] = rstpoint[2].v;
    rst_buffer.T2[0] = 0;
    rst_buffer.T2[1] = rstpoint[5].u;
    rst_buffer.T2[2] = rstpoint[5].v;
    rst_buffer.S2[0] = 0;
    rst_buffer.S2[1] = rstpoint[4].u;
    rst_buffer.S2[2] = rstpoint[4].v;
    rst_buffer.R2[0] = 0;
    rst_buffer.R2[1] = rstpoint[3].u;
    rst_buffer.R2[2] = rstpoint[3].v;
    rst_buffer.t = transfer_buffer.Time;
    rst_buffer.status = transfer_buffer.status;
    if (MvsFile)
    {
       fprintf(mvs,"rst_buffer.t = %f\n",rst_buffer.t);
    /* fprintf(mvs,"rst_buffer.R1: %f,  %f,
%f\n",rst_buffer.R1[0],rst_buffer.R1[1],rst_buffer.R1[2]);
       fprintf(mvs,"rst_buffer.S1: %f,  %f,
%f\n",rst_buffer.S1[0],rst_buffer.S1[1],rst_buffer.S1[2]);
       fprintf(mvs,"rst_buffer.T1: %f,  %f,
%f\n",rst_buffer.T1[0],rst_buffer.T1[1],rst_buffer.T1[2]);
       fprintf(mvs,"rst_buffer.R2: %f,  %f,
%f\n",rst_buffer.R2[0],rst_buffer.R2[1],rst_buffer.R2[2]);
       fprintf(mvs,"rst_buffer.S2: %f,  %f,
%f\n",rst_buffer.S2[0],rst_buffer.S2[1],rst_buffer.S2[2]);
       fprintf(mvs,"rst_buffer.T2: %f,  %f,
%f\n",rst_buffer.T2[0],rst_buffer.T2[1],rst_buffer.T2[2]);
    */
    }
}
```

Module Scan.h

This module is a header file for the module scan.c

Source code

```
/*******************************************************
Filename: SCAN.h
*******************************************************/

unsigned char TurnLaserOn(void);
unsigned char TurnLaserOff(void);
unsigned char StartVision(void);
unsigned char StopVision(void);
unsigned char SyncronizeTime(void);
unsigned char SetTemplate(unsigned char TempName[79]);
unsigned char ReadRSTPoint(void);

void ExtractRSTPoints();
```

Module Gpibdrv.c

This module is driver software for a low level GPIB-interface between the user written software procedures and procedures on an external hardware platform. The laser range sensor is operated by the external Platform. The software is a commercial source code, which is adapted to the specific control system and hardware addresses. Since there is a copy right on the commercial source code it is not provided in this appendix. Another reason is that only a minor part of this source code is actually produced within this specific work.

Module Gpibdrv.h

This module is a header file for the above mentioned GPIB interface, and is not provided in this appendix.

Module Queue.c

This module controls the queue of robot locations. The robot location is updated with a relatively high fequency in order to estimate the position of the sensor at the time measurement are made. All operations on the queue of robot locations are controlled by this module.

Source code

```
/*****************************************************************************
Filename:   queue.c

                    Module for handling of queue operations.

                    Author Henrik John Andersen

                    january 5 1998.
*****************************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <queue.h>
#include <global.h>
#include <matlib.h>

char QueueInitialized = FALSE;
char QueueFile = TRUE, measposFile = FALSE;
double NumberOfQueueElements;
Node *head, *current;
FILE *qfile, *measpos;

/************************************/


void InitializeQueue(void)
{
    head   = NULL;
    current= NULL;
    if(QueueFile == TRUE)
    {
        NumberOfQueueElements = 0;
        qfile  = fopen("qfile.log","w+");
        fprintf(qfile,"QFILE IS OPENED\n");
    }
    if(measposFile == TRUE)
    {
        measpos = fopen("measpos.log","w+");
        fprintf(measpos,"measpos IS OPENED\n");
        fprintf(measpos,"Denne fil indeholder maalte positioner, som robotten har
gennemloebet\n");
    }

}
/************************************/

void InsertNode(void)
{
    Node *NewNode;

    /* allocate memory for node */
    if ((NewNode = (Node *) malloc(sizeof(Node))) == NULL)
    {
        if(QueueFile == TRUE)
        {
            fprintf(qfile,"Not enough memory to allocate buffer\n");
        }
    }
    NumberOfQueueElements++;
    if(QueueFile == TRUE)
    {
        fprintf(qfile,"Insert Node; NumberOfQueueElements: %.0lf\n",
NumberOfQueueElements);
    }
```

```
    NewNode->next = head;
    head = NewNode;
    current = head;
}


/************************************/

void UpdateNode(element el)
{
    current->Element = el;
    if(QueueFile == TRUE)
        fprintf(qfile,"Node is Updated at time = %.2lf \n\n", current->Element.time);
}


/************************************/

void DeleteNode(void)       /* Deletes current */
{

    Node *IntermediateNode;

    if(current != head)
    {
        IntermediateNode = head;
        while (IntermediateNode->next != current)
            IntermediateNode = IntermediateNode->next;
    }
    else head = head->next;
    free(current);
    current = head;
    NumberOfQueueElements--;
}


/************************************/

void DeleteEndOfQueue(void)   /* Deletes all nodes behind current */
{

    Node *dummy;

    if(current->next != NULL)
    {
        dummy = current->next;
        current->next = NULL;
        current = dummy;
        while(current->next != NULL)
        {
            dummy = current->next;
            current->next = NULL;
            free(current);
            NumberOfQueueElements--;
            current = dummy;
            if(QueueFile == TRUE)
            {
                fprintf(qfile,"Node is deleted\n");
                fprintf(qfile,"NumberOfQueueElements: %.0lf\n", NumberOfQueueElements);
            }

        }
        free(current);
        NumberOfQueueElements--;
```

```
        if(QueueFile == TRUE)
        {
            fprintf(qfile,"Node is deleted\n");
            fprintf(qfile,"NumberOfQueueElements: %.0lf\n", NumberOfQueueElements);
        }
    }
}

/*************************************/

void ResetRobotLocationQueue(void)   /* Deletes all nodes in the queue */
{
    Node *IntermediateNode;

    current = head;

    while (current != NULL)
    {
        if(current->next != NULL)
        {
            IntermediateNode = (Node *)current->next;
            current->next = NULL;
        }
        else IntermediateNode = NULL;
        free(current);
        if(IntermediateNode != NULL)
            current = IntermediateNode;
        else
            current = NULL;
        NumberOfQueueElements--;
        if(QueueFile == TRUE)
            fprintf(qfile,"NumberOfQueueElements = %.0lf\n", NumberOfQueueElements);
    }
    NumberOfQueueElements = 0;
    head = NULL;
    current =   NULL;
    QueueInitialized = FALSE;
    if(QueueFile == TRUE)
    {
        fprintf(qfile,"RobotLocationQueue is Resat\n\n");
        fprintf(qfile,"QFILE IS CLOSED");
        fclose(qfile);
    }
    if(measposFile == TRUE)
    {
        fprintf(measpos,"measpos IS CLOSED");
        fclose(measpos);
    }

}

/*************************************/

/*Sets current to the node containing the robot position right after the
sensor measurement.  current->next points to the robot position taken right
before (or at the same time as) the sensor measurement. */

int FindNearestNodes(double SensorClock)
{
    Node *dummy;
```

```
    if(head->Element.time < SensorClock)
    {
        if(QueueFile == TRUE)
        {
            fprintf(qfile,"head->Element.time = %.3lf\n",head->Element.time);
            fprintf(qfile,"SensorClock = %.3lf\n",SensorClock);
            fprintf(qfile,"actual SensorClock is in front of last updated robot
position\n");
        }
        current = NULL;
        fflush(qfile);
        return -1;
    }

    current = head;
    dummy = (Node *)head->next;

    while(dummy->Element.time > SensorClock)
    {
        current = dummy;
        if(current->next == NULL)
        {
            if(QueueFile == TRUE)
                fprintf(qfile,"The sensor measurement is older than oldest robot
location\n");
                fflush(qfile);
            return -2;
        }

        else
            dummy = (Node *)current->next;
    }
    return 0;
}


/*************************************/


void tryUpdateRobotLoc(double T, double X, double Y, double Z, double B, double C)
{
    element NewElement;
    Point Previous, Actual;
    float UpdateFrequency = 2;    /* Hz , Must not be 0! */
    char TimeDummy;
    if (QueueInitialized == FALSE)
    {
        InitializeQueue();
        Previous[0] = 0;
        Previous[1] = 0;
        Previous[2] = 0;
    }
    else
    {
        Previous[0] = head->Element.x;
        Previous[1] = head->Element.y;
        Previous[2] = head->Element.z;
    }
    Actual[0] = X;
    Actual[1] = Y;
    Actual[2] = Z;

    if(measposFile == TRUE)
    {
        fprintf(measpos,"Actual_X = %.1f; Actual_Y = %.1f; Actual_Z = %.1f;\n",
        X/1000, Y/1000, Z/1000);
    }
```

```
    if ((QueueInitialized == TRUE) &&
        (T - head->Element.time >= 1/UpdateFrequency))
        TimeDummy = TRUE;
    else TimeDummy = FALSE;


    if ((QueueInitialized == FALSE) || (distance(&Previous, &Actual)  > 50) ||
        (TimeDummy == TRUE))
    {
        InsertNode();
        NewElement.time = T;
        NewElement.x = X;
        NewElement.y = Y;
        NewElement.z = Z;
        NewElement.b = B;
        NewElement.c = C;

        UpdateNode(NewElement);
        QueueInitialized = TRUE;
    }
}

/************************************/


/* This function is only feasable for finding the robotlocation at a certain
       point in time. As input the function requires a pointer to the queue of
       robotlocations, the time for which the position is wanted, and pointers
       to the addresses of X, Y, Z, B and C. */

int tryFindRobotPosition(double Tsensor, double *LocalXPos,
    double *LocalYPos, double *LocalZPos, double *LocalBPos, double *LocalCPos)
{
    Node *before;

    int dummy;
    if(QueueFile == TRUE)
        fprintf(qfile,"finding robotposition and goto FindNearestNodes\n",*(LocalXPos));

    dummy = FindNearestNodes(Tsensor);
    if (dummy == -2)
        return FALSE;

    if(dummy == -1)
        return FALSE;

    before = (Node *)current->next;
    if(Tsensor == before->Element.time)
    {
        *(LocalXPos) = before->Element.x;
        *(LocalYPos) = before->Element.y;
        *(LocalZPos) = before->Element.z;
        *(LocalBPos) = before->Element.b;
        *(LocalCPos) = before->Element.c;
    }

    else if(Tsensor == current->Element.time)
    {
        *(LocalXPos) = current->Element.x;
        *(LocalYPos) = current->Element.y;
        *(LocalZPos) = current->Element.z;
        *(LocalBPos) = current->Element.b;
        *(LocalCPos) = current->Element.c;
    }
```

```
    else if((Tsensor > before->Element.time) &&
                    (Tsensor < current->Element.time))
    {
        *(LocalXPos) = before->Element.x +
        (current->Element.x - before->Element.x)/
        (current->Element.time -  before->Element.time) *
        (Tsensor - before->Element.time);

        *(LocalYPos) = before->Element.y +
        (current->Element.y - before->Element.y)/
        (current->Element.time -  before->Element.time) *
        (Tsensor - before->Element.time);

        *(LocalZPos) = before->Element.z +
        (current->Element.z - before->Element.z)/
        (current->Element.time -  before->Element.time) *
        (Tsensor - before->Element.time);

        *(LocalBPos) = before->Element.b +
        (current->Element.b - before->Element.b)/
        (current->Element.time -  before->Element.time) *
        (Tsensor - before->Element.time);

        *(LocalCPos) = before->Element.c +
        (current->Element.c - before->Element.c)/
        (current->Element.time -  before->Element.time) *
        (Tsensor - before->Element.time);
    }
        current = before;
        DeleteEndOfQueue();
    current = head;
    if(QueueFile == TRUE)
    {
        fprintf(qfile,"Time: %lf\n",Tsensor);
        fprintf(qfile,"Xpos: %lf\n",*(LocalXPos));
    }
    return TRUE;
}

/************************************/
```

Module Queue. h

This module is the header file for module “Queue.c”

Source code

```
/*****************************************************************************
Filename:   queue.h
                    Header file for the queue.c module.
*****************************************************************************/

/*#include <global.h>*/

typedef struct{
double x, y, z, b, c, time;
}element;


typedef struct{
element  Element;
void  *next;
}Node;
```

```
typedef struct{
Node *head, *current;
}Queue;

/*
typedef struct{
RST_buffer RST_element;
}rst_element;


typedef struct{
rst_element  RstElement;
void  *next;
}RstNode;

typedef struct{
RstNode *head, *current;
}RstQueue;

*/


void InitializeQueue(void);
void InsertNode(void);
void UpdateNode(element el);
void DeleteNode(void);
void DeleteEndOfQueue(void);   /* Deletes current and all nodes behind this */
void ResetRobotLocationQueue(void);
void tryUpdateRobotLoc(double T, double X, double Y, double Z, double B, double C);
int  tryFindRobotPosition(double Tsensor, double *LocalXPos,
          double *LocalYPos, double *LocalZPos, double *LocalBPos, double *LocalCPos);
```

Module Matlib.c

This module is a library for mathematic functions used in the system. Some of the procedures are taken directly from existing software modules in other systems.

Source code:

```
/*****************************************************************************
* Filename      : MatLib.C                              Date: 05.07.1996/OM
*****************************************************************************
*
* Author : Ole Madsen
*
* Description  : Mathematic functions used in the search system and the
*                  welding controller.
*
*****************************************************************************/
#include <stdio.h>
#include <global.h>
#include <math.h>
#include <matlib.h>


const double Err  = 0.00001;
const double Cerr = 0.000000001;
```

```
/*-------------------------------------------------------------------------
/*   Geometric procedures and functions concerning planes in space
/*-------------------------------------------------------------------------*/


/*-------------------------------------------------------------------------
/* void makePlane(Point p1, Point p2, Point p3, double *PL)
/*-------------------------------------------------------------------------
/* Returns the four parameters (A,B,C and D) of the equation for a plane:
/*                  Ax + By + Cz  = D
/* The inputs are three points in the plane and an address for the plane.
/*-------------------------------------------------------------------------*/
void makePlane(Point p1, Point p2, Point p3, double *PL)
{
   int i;
   double maxlength, nextlength, dummylength;
   Vector maxvec, nextvec, dummyvec, NormalVector;

   /** We find the two longest vectors in order to minimize uncertainties) **/

   makeVector(p1,p2,&maxvec[0]);
   makeVector(p1,p3,&nextvec[0]);

   /*maxlength = vectorLength(maxvec);
   nextlength = vectorLength(nextvec);
   if(nextlength > maxlength)
   {
      dummylength = maxlength;
      *dummyvec = *maxvec;
      maxlength = nextlength;
      *maxvec = *nextvec;
      nextlength = dummylength;
      *nextvec = *dummyvec;
   }
   makeVector(p2,p3,&dummyvec[0]);
   dummylength = vectorLength(dummyvec);

   if (dummylength > maxlength)
   {
      nextlength = maxlength;
      *nextvec = *maxvec;
      maxlength = dummylength;
      *maxvec = *dummyvec;
   }
   else if (dummylength > nextlength)
   {
      nextlength = dummylength;
      *nextvec = *dummyvec;
   } */

   cross(maxvec,nextvec,&NormalVector[0]);

   *PL = NormalVector[0];
   *(PL+1) = NormalVector[1];
   *(PL+2) = NormalVector[2];
   *(PL+3) = 0;
   for (i=0;i<3;i++)
      *(PL+3) = *(PL+3) + NormalVector[i] * (*(p1+i));
}
```

```
/*-------------------------------------------------------------------------
/* double distBetweenPointAndPlane(Plane Pl, Point P)
/*-------------------------------------------------------------------------
/* Returns the distance between a point and a plane in space:
/* The inputs are a plane and a point in space.
/*-------------------------------------------------------------------------*/
double distBetweenPointAndPlane(Plane Pl, Point P)
{
double dist;
dist = Pl[0] * P[0] +Pl[1] * P[1] +Pl[2] * P[2] - Pl[3];
dist = dist/(sqrt((Pl[0]*Pl[0])+(Pl[1]*Pl[1])+(Pl[2]*Pl[2])));
if (dist < 0)
   dist = dist * (-1);
return dist;
}


/*-------------------------------------------------------------------------
/*   Procedures and functions involving lines and points in two dimensions
/*-------------------------------------------------------------------------*/


/*-------------------------------------------------------------------------
/* void makeLine(double A1, double A2,  double B1, double B2, double *a, double *b,
double *c)
/*-------------------------------------------------------------------------
/* Returns the three parameters (a,b and c) of the equation for a line:
/*                  ax + by + C  = 0     (For two dimensional coordinate system)
/* The inputs are two points (A1,A2) and (B1,B2) in a two dimensional coor-
/* dinate system, and an address for the parameters a, b and c.
/*-------------------------------------------------------------------------*/


void makeLine(double A1, double A2,  double B1, double B2, double *a, double *b, double
*c)
{
   *a = (B2 - A2)/(B1 - A1);
   *b = -1;
   *c = A2 - ((*a) * A1);
}


/*-------------------------------------------------------------------------
/* double distBetweenPointAndLine(double P1, double P2, double a, double b, double c)
/*-------------------------------------------------------------------------
/* Returns the distance between a point (P) and a line (aX + bY + c = 0)
/* in a two dimensional Coordinate system:
/* The inputs are the line parameters (a, b and c) and a two dimensional
/* point (P).
/*-------------------------------------------------------------------------*/
double distBetweenPointAndLine(double P1, double P2, double a, double b, double c)
{
double dist;

dist = (a*P1 + b*P2 + c)/(sqrt(sqr(a) + sqr(b)));

if (dist < 0)
   dist = dist * (-1);
return dist;
}
```

```
/*-------------------------------------------------------------------------
/* void IntersectionBetweenLines(double a1, double b1, double c1, double a2, double b2,
double c2, double *P1, double *P2)
/*-------------------------------------------------------------------------
/* Returns the point (P1, P2) of intersection between a line (a1X + b1Y + c1 = 0)
/* and a line (a2X + b2Y + c2 = 0)
/* in a two dimensional Coordinate system:
/* The inputs are the line parameters (a1, b1, c1, a2, b2, c2) and the address of
/* the two dimensional point (P).
/*-------------------------------------------------------------------------*/
void IntersectionBetweenLines(double a1, double b1, double c1, double a2, double b2,
double c2, double *P1, double *P2)
{
*P2 = (a1*c2/a2 - c1)/(b1 - a1*b2/a2);
*P1 = -c2/a2 - b2/a2 * (*P2);
}



/*-------------------------------------------------------------------------
/*   Vector procedures and functions
/*-------------------------------------------------------------------------*/


/*-------------------------------------------------------------------------
/* double dot(Vector A, Vector B);
/*-------------------------------------------------------------------------
/* Returns the dot product of A and B.
/*-------------------------------------------------------------------------*/

double dot(Vector A, Vector B) {
   return(A[0]*B[0]+A[1]*B[1]+A[2]*B[2]);
}

/*-------------------------------------------------------------------------
/* double distance(Point P1,Point P2);
/*-------------------------------------------------------------------------
/* Returns the distance between two points P1 and P2.
/*-------------------------------------------------------------------------*/

double distance(Point *P1, Point *P2) {
   int i;
   double sum = 0;
   for (i=0;i<3;i++)
    sum += (P2[i]-P1[i])*(P2[i]-P1[i]);
   return sqrt(sum);
}

/*-------------------------------------------------------------------------
/* double vectorLength(Vector V);
/*-------------------------------------------------------------------------
/* Returns the length of a vector
/*-------------------------------------------------------------------------*/

double vectorLength(Vector V) {
   int i;
   double sum=0;
   for (i=0;i<3;i++)
    sum += (V[i])*(V[i]);
   return(sqrt(sum));
}

/*-------------------------------------------------------------------------
/* double AngleBetweenVectors(Vector V1, Vector V2)
/*-------------------------------------------------------------------------
/* Returns the acute (spidse) angle between the vectors V1 and V2
```

```
/*-------------------------------------------------------------------------*/

double AngleBetweenVectors(Vector V1, Vector V2)
{
   double Angle;
   Angle = acos(dot(V1,V2)/(vectorLength(V1)*vectorLength(V2)));
   Angle = Angle * 180/pi;
   return (Angle);
}

/*-------------------------------------------------------------------------
/* void cross(Vector A,Vector B,double *C);
/*-------------------------------------------------------------------------
/* Computes the cross product of A and B. The result is contained in the
/* addresses of a vector type, starting with the element C at the initial
/* address.
/*-------------------------------------------------------------------------*/

void cross(Vector A,Vector B,double * const C) {

   *(C) =   A[1]*B[2]-A[2]*B[1];
   *(C+1) = A[2]*B[0]-A[0]*B[2];
   *(C+2) = A[0]*B[1]-A[1]*B[0];
}



/*-------------------------------------------------------------------------
/* bool unitVector(Vector A, double *e);
/*-------------------------------------------------------------------------
/* Computes the unit vector of A. The result is contained in a vector type,
/* starting with the element e at the initial address.
/* If the length of A = 0 then the function will return 0
/* else it will return 1
/*-------------------------------------------------------------------------*/
int unitVector(Vector A, double * const e) {
   double l = vectorLength(A);
   if (l < Err){
    return(0);
   }

   *e = A[0]/l;
   *(e + 1) = A[1]/l;
   *(e + 2) = A[2]/l;

   return(1);
}
/*-------------------------------------------------------------------------
/* void makeVector(Point P1,Point P2, double * const V);
/*-------------------------------------------------------------------------
/* Computes the vector between P1 and P2. The result is contained in a vector
/* type, starting with the element V in the initial address.
/*-------------------------------------------------------------------------*/

void makeVector(Point P1,Point P2, double * const V){
   int i;
   for (i=0;i<3;i++)
    *(V + i) = P2[i]-P1[i];
}
```

```
/*-------------------------------------------------------------------------
/*              TransformationMatrix procedures
/*-------------------------------------------------------------------------*/


/*-------------------------------------------------------------------------
/* void Invers(TransformationMatrix Txu,TransformationMatrix &Tux);
/*-------------------------------------------------------------------------
/* Computes the invers of transformationMatrix Txu. The result in contrined
/* in Tux.
/*-------------------------------------------------------------------------*/

void Invers(TransformationMatrix Txu,double * const Tux) {
  TransformationMatrix T;
  int i,j,dummy=0;
  for (i=0;i<3;i++) {
     for (j=0;j<3;j++)
        T[j][i]=Txu[i][j];}
  for (i=0;i<3;i++)
     T[i][3] = -(Txu[0][i]*Txu[0][3]+Txu[1][i]*Txu[1][3]+Txu[2][i]*Txu[2][3]);
  for (i=0;i<3;i++)
     for (j=0;j<4;j++){
    *(Tux + dummy++) = T[i][j];
     }
}

/*-------------------------------------------------------------------------
/* void matrixProd(TransformationMatrix T1,
          TransformationMatrix T2,
          double * const res) {
/*-------------------------------------------------------------------------
/* Computes the multiplum of T1 and T2. The result is contained in res.
/*-------------------------------------------------------------------------*/


void matrixProd(TransformationMatrix T1,
       TransformationMatrix T2,
       double * const res) {
  int i,j,MatrixDummy = 0;
  TransformationMatrix T;

  for (i=0;i<3;i++)
     for (j=0;j<3;j++)
     T[i][j] = T1[i][0]*T2[0][j]+T1[i][1]*T2[1][j]+T1[i][2]*T2[2][j];

  for (i=0;i<3;i++)
    T[i][3] = T1[i][0]*T2[0][3]+T1[i][1]*T2[1][3]+T1[i][2]*T2[2][3]+T1[i][3];

  for (i=0;i<3;i++)
     for (j=0;j<4;j++)
        *(res+MatrixDummy++) = T[i][j];
}


/*-------------------------------------------------------------------------
/*
/*void pointTransformation(TransformationMatrix T,Point P1,double * const P2)
/*
/*-------------------------------------------------------------------------
/* Transforms a point P1 from a representation relatively to a frame a to
/* to a representation relatively to a frame b. The transformation from
/* frame b to frame a (bTa)is represented by the transformation matrix T.
/* Pb = bTa * Pa.
/*-------------------------------------------------------------------------*/

void pointTransformation(TransformationMatrix T,Point P1,double * const P2){
  Point P;
```

```
  int i;

  for (i=0;i<3;i++)
     P[i]=T[i][0]*P1[0]+T[i][1]*P1[1]+T[i][2]*P1[2]+T[i][3];
  for (i=0;i<3;i++)
   *(P2+i) = P[i];
}

/*-------------------------------------------------------------------------
/*
/*void vectorTransformation(TransformationMatrix R,Vector A,double * const B)
/*
/*-------------------------------------------------------------------------
/* Transforms a vector A from a representation relatively to a frame a to
/* to a representation relatively to a frame b.The transformation from
/* frame b to frame a (bTa)is represented by the transformation matrix T.
/* Vb = bTa * Va.
/*-------------------------------------------------------------------------*/


void vectorTransformation(TransformationMatrix R,Vector A,double * const B){
    Vector V;
    int i,j;
    for (i=0;i<3;i++)
     { double sum=0;
       for (j=0;j<3;j++)
       sum+=A[j]*R[i][j];
       V[i] = sum;
     }
    for (i=0;i<3;i++)
    *(B+i) = V[i];
}


double sqr(double X)
{ return(X*X);}

/*-------------------------------------------------------------------------}
/* void transformABCToTransMatrix(Point P,
/*                  double A,double B,double C,
/*                  TransformationMatrix &Tt);
/*-------------------------------------------------------------------------}
/* From Craig pp. 45                                                        }
/*-------------------------------------------------------------------------*/

/*
void transformABCToTransMatrix(Point P,
                  double A,double B,double C,
                  TransformationMatrix &Tt)
{
  Tt[0][0] = cos(A)*cos(B);
  Tt[1][0] = sin(A)*cos(B);
  Tt[2][0] =-sin(B);

  Tt[0][1] = cos(A)*sin(B)*sin(C)-sin(A)*cos(C);
  Tt[1][1] = sin(A)*sin(B)*sin(C)+cos(A)*cos(C);
  Tt[2][1] = cos(B)*sin(C);

  Tt[0][2] = cos(A)*sin(B)*cos(C)+sin(A)*sin(C);
  Tt[1][2] = sin(A)*sin(B)*cos(C)-cos(A)*sin(C);
  Tt[2][2] = cos(B)*cos(C);

  Tt[0][3] = P[0];
  Tt[1][3] = P[1];
  Tt[2][3] = P[2];
}
*/
```

```
/*
void transformTransmatrixToABC(TransformationMatrix Tt,
                  double &A, double &B, double &C)
{

  B = atan2(-Tt[2][0],sqrt(sqr(Tt[0][0])+sqr(Tt[1][0])));
  if (fabs(B-pi/2)>Cerr) {
    A = atan2(Tt[1][0],Tt[0][0]);
    C = atan2(Tt[2][1],Tt[2][2]);
    return;
  }
  A = atan2(-Tt[0][1],Tt[1][1]);
  C = 0;
}
*/
```

Module Matlib.h

This module is the header file for module “matlib.c”.

Source code

```
#define pi 3.141592653589793
#define RadToDegree=180/pi;
#define DegreeToRad = pi/180;

typedef double TransformationMatrix[3][4];

typedef double Vector[3];
typedef double Plane[4];
/*
class polyCurve {
   public:
    double a0,a1,a2,a3;
    int order;
    double calcPoly(double Yi);
    Boolean Intersection(polyCurve B,Point *P);
};
*/

/* Vector procedures and functions */
void makeLine(double A1, double A2,  double B1, double B2, double *a, double *b, double
*c);

double distBetweenPointAndLine(double P1, double P2, double a, double b, double c);

void IntersectionBetweenLines(double a1, double b1, double c1, double a2, double b2,
double c2, double *P1, double *P2);

void makePlane(Vector V1, Vector V2, Point P, double *PL);

double distBetweenPointAndPlane(Plane Pl, Point P);

double dot(Vector A, Vector B);

double distance(Point *P1, Point *P2);

double vectorLength(Vector V);

double AngleBetweenVectors(Vector V1, Vector V2);

void cross(Vector A,Vector B,double * const C);

int unitVector(Vector A, double * const e);
```

```
void makeVector(Point P1,Point P2, double * const V);


/* TransformationMatrix procedures */


void Invers(TransformationMatrix Txu,double * const Tux);

void matrixProd(TransformationMatrix T1,
       TransformationMatrix T2,
       double * const res);

void pointTransformation(TransformationMatrix T,Point P1,double * const P2);

void vectorTransformation(TransformationMatrix R,Vector A,double * const B);

double sqr(double X);

/*
void transformABCToTransMatrix(Point P,
                  double A,double B,double C,
                  TransformationMatrix &Tt);

void transformTransmatrixToABC(TransformationMatrix Tt,
                  double &A, double &B, double &C);
*/
```

Appendix D

Experiments

This appendix shows the documentation of experiments described in chapter 7. The type of base material and dimensions, filler material and wire thickness, protection gas, and plasma control gas were the same for all experiments.

The base material was:

Thickness of workpieces was 10 mm for both workpieces in the joint.

Filler material was: ok autrod 12.51.

Thickness of filler wire was 1.0 mm.

Type of protection gas and plasma control gas were Helium.

Experiment ID: 6B5	Section A: *Binding depth:* *4.5 mm* *Penetration depth:* *7.1 mm* *Notch size:* *0.2 mm* *Curve length:* *50 mm* Section B: *Binding depth:* *5.1 mm* *Penetration depth:* *8.0 mm* *Notch size:* *0.3 mm* *Curve length:* *180 mm* Section C: *Binding depth:* *4.5 mm* *Penetration depth:* *7.7 mm* *Notch size:* *0.3 mm* *Curve length:* *250 mm*	
Corner radius: 0.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.6 kW		
Binding depth: 4.7 mm (average)		
Penetration depth: 7.6 mm (average)		
Notch size: 0.2 mm (average)		
Stiffener from profile number: 6		
Surface cracks except start/end: Yes		
Remarks: Normal conditions		

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	900	400
0.3	0,5	750	500
0.6	0,7	750	900
0.9	1,1	700	1650

Experiment ID: 6B6	Section A: *Binding depth:* *5.1 mm* *Penetration depth:* *7.2 mm* *Notch size:* *0.3 mm* *Curve length:* *180 mm* Section B: *Binding depth:* *5.2 mm* *Penetration depth:* *7.3 mm* *Notch size:* *0.2 mm* *Curve length:* *280 mm* Section C: *Binding depth:* *4.6 mm* *Penetration depth:* *7.3 mm* *Notch size:* *0.2 mm* *Curve length:* *360 mm*	
Corner radius: 0.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.6 kW		
Binding depth: 5.0 mm (average)		
Penetration depth: 7.3 mm (average)		
Notch size: 0.2 mm (average)		
Stiffener from profile number: 6		
Surface cracks except start/end: No		
Remarks: Normal conditions		

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	900	400
0.3	0,5	750	500
0.6	0,7	750	900
0.9	1,1	700	1650

Experiment ID: 6B7	Section A: *Binding depth:* *5.1 mm* *Penetration depth:* *7.4 mm* *Notch size:* *0.2 mm* *Curve length:* *100 mm* Section B: *Binding depth:* *5.0 mm* *Penetration depth:* *6.9 mm* *Notch size:* *0.3 mm* *Curve length:* *230 mm* Section C: *Binding depth:* *4.2 mm* *Penetration depth:* *7.4 mm* *Notch size:* *0.1 mm* *Curve length:* *380 mm*	
Corner radius: 0.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.5 kW		
Binding depth: 4.8 mm (average)		
Penetration depth: 7.2 mm (average)		
Notch size: 0.2 mm (average)		
Stiffener from profile number: 6		
Surface cracks except start/end: Yes		
Remarks: Normal conditions		

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	900	400
0.3	0,5	750	500
0.6	0,7	750	900
0.9	1,1	700	1650

Experiment ID: 6B8	Section A: *Binding depth:* *5.4 mm* *Penetration depth:* *7.8 mm* *Notch size:* *0.3 mm* *Curve length:* *130 mm* Section B: *Binding depth:* *5.2 mm* *Penetration depth:* *8.1 mm* *Notch size:* *0.2 mm* *Curve length:* *270 mm* Section C: *Binding depth:* *5.5 mm* *Penetration depth:* *7.7 mm* *Notch size:* *0.2 mm* *Curve length:* *380 mm*	
Corner radius: 0.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.5 kW		
Binding depth: 5.4 mm (average)		
Penetration depth: 7.9 mm (average)		
Notch size: 0.2 mm (average)		
Stiffener from profile number: 6		
Surface cracks except start/end: Yes		
Remarks: Normal conditions		

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	900	400
0.3	0,5	750	500
0.6	0,7	750	900
0.9	1,1	700	1650

Experiment ID: 6B10	Section A: *Binding depth:* *6.2 mm* *Penetration depth:* *8.0 mm* *Notch size:* *0.7 mm* *Curve length:* *110 mm* Section B: *Binding depth:* *6.8 mm* *Penetration depth:* *9.5 mm* *Notch size:* *0.5 mm* *Curve length:* *200 mm* Section C: *Binding depth:* *3.0 mm* *Penetration depth:* *3.7 mm* *Notch size:* *0.2 mm* *Curve length:* *360 mm*	
Corner radius: 0.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.5 mm	*Measured Gap:*	
Laser Power at weld scene: 7.5 Kw		
Binding depth: 6.5 mm (average)		
Penetration depth: 8.8 mm (average)		
Notch size: 0.6 mm (average)		
Stiffener from profile number: 6		
Surface cracks except start/end: No		

Remarks:

Experiment with gap size 0.5 mm.

Section 6B10C is not considered in the interpretation of experiments, because it is regarded as a result of equipment drop-out.

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	900	400
0.3	0,5	750	500
0.6	0,7	750	900
0.9	1,1	700	1650

Experiment ID: 71E1	Section A: *Binding depth:* *5.0 mm* *Penetration depth:* *7.3 mm* *Notch size:* *0.3 mm* *Curve length:* *230 mm* Section B: *Binding depth:* *4.9 mm* *Penetration depth:* *7.8 mm* *Notch size:* *0.3 mm* *Curve length:* *200 mm*	
Corner radius: 0.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 0 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.5 Kw		
Binding depth: 4.9 mm (average)		
Penetration depth: 7.6 mm (average)		
Notch size: 0.3 mm (average)		
Stiffener from profile number: 2		
Surface cracks except start/end: ?		
Remarks: Plasma control gas flow is set to zero		

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	900	400
0.3	0,5	750	500
0.6	0,7	750	900
0.9	1,1	700	1650

Experiment ID: 71E2	Section A: *Binding depth:* *4.7 mm* *Penetration depth:* *8.1 mm* *Notch size:* *0.1 mm* *Curve length:* *230 mm* Section B: *Binding depth:* *4.4 mm* *Penetration depth:* *8.6 mm* *Notch size:* *0.2 mm* *Curve length:* *200 mm*	
Corner radius: 0.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 16 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.5 kW		
Binding depth: 4.6 mm (average)		
Penetration depth: 8.3 mm (average)		
Notch size: 0.2 mm (average)		
Stiffener from profile number: 2		
Surface cracks except start/end: ?		
Remarks: Plasma control gas flow is increased		

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	900	400
0.3	0,5	750	500
0.6	0,7	750	900
0.9	1,1	700	1650

Experiment ID: 71E3	Section A: *Binding depth:* *5.0 mm* *Penetration depth:* *8.2 mm* *Notch size:* *0.2 mm* *Curve length:* *230 mm* Section B: *Binding depth:* *4.5 mm* *Penetration depth:* *9.2 mm* *Notch size:* *0.2 mm* *Curve length:* *200 mm*	
Corner radius: 0.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 29 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.5 kW		
Binding depth: 4.7 mm (average)		
Penetration depth: 8.7 mm (average)		
Notch size: 0.2 mm (average)		
Stiffener from profile number: 2		
Surface cracks except start/end: ?		
Remarks: Plasma control gas flow is increased		

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	900	400
0.3	0,5	750	500
0.6	0,7	750	900
0.9	1,1	700	1650

Experiment ID: 72B1	Section A: *Binding depth:* *5.2 mm* *Penetration depth:* *7.2 mm* *Notch size:* *0.5 mm* *Curve length:* *230 mm* Section B: *Binding depth:* *5.0 mm* *Penetration depth:* *7.6 mm* *Notch size:* *0.5 mm* *Curve length:* *200 mm*	
Corner radius: 2.5 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.45 kW		
Binding depth: 5.1 mm (average)		
Penetration depth: 7.4 mm (average)		
Notch size: 0.5 mm (average)		
Stiffener from profile number: 3		
Surface cracks except start/end: Yes		

Remarks:

Corner radius increased and measured gap overwritten by 0.0 mm in the system

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	900	400
0.3	0,5	750	500
0.6	0,7	750	900
0.9	1,1	700	1650

Experiment ID: 72B2	Section A: *Binding depth:* *4.8 mm* *Penetration depth:* *7.4 mm* *Notch size:* *0.5 mm* *Curve length:* *230 mm* Section B: *Binding depth:* *4.7 mm* *Penetration depth:* *8.1 mm* *Notch size:* *0.4 mm* *Curve length:* *200 mm*	
Corner radius: 2.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.5 kW		
Binding depth: 4.7 mm (average)		
Penetration depth: 7.7 mm (average)		
Notch size: 0.5 mm (average)		
Stiffener from profile number: 3		
Surface cracks except start/end: No		
Remarks: Corner radius increased and measured gap overwritten by 0.0 mm in the system		

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	900	400
0.3	0,5	750	500
0.6	0,7	750	900
0.9	1,1	700	1650

Experiment ID: 72B3	Section A: *Binding depth:* *4.3 mm* *Penetration depth:* *5.7 mm* *Notch size:* *0.1 mm* *Curve length:* *230 mm* Section B: *Binding depth:* *4.6 mm* *Penetration depth:* *6.5 mm* *Notch size:* *0.5 mm* *Curve length:* *200 mm*	
Corner radius: 1.5 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.45 kW		
Binding depth: 4.5 mm (average)		
Penetration depth: 6.1 mm (average)		
Notch size: 0.3 mm (average)		
Stiffener from profile number: 3		
Surface cracks except start/end: No		
Remarks: Corner radius increased and measured gap overwritten by 0.0 mm in the system		

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	900	400
0.3	0,5	750	500
0.6	0,7	750	900
0.9	1,1	700	1650

Experiment ID: 72B4	Section A: *Binding depth:* *4.4 mm* *Penetration depth:* *7.2 mm* *Notch size:* *0.2 mm* *Curve length:* *280 mm* Section B: *Binding depth:* *4.5 mm* *Penetration depth:* *7.2 mm* *Notch size:* *0.2 mm* *Curve length:* *250 mm*	
Corner radius: 1.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.45 kW		
Binding depth: 4.5 mm (average)		
Penetration depth: 7.2 mm (average)		
Notch size: 0.2 mm (average)		
Stiffener from profile number: 3		
Surface cracks except start/end: No		
Remarks: Corner radius increased and measured gap overwritten by 0.0 mm in the system		

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	900	400
0.3	0,5	750	500
0.6	0,7	750	900
0.9	1,1	700	1650

Experiment ID: 73B1	Section A: *Binding depth:* *6.5 mm* *Penetration depth:* *7.1 mm* *Notch size:* *0.4 mm* *Curve length:* *230 mm* Section B: *Binding depth:* *7.1 mm* *Penetration depth:* *8.0 mm* *Notch size:* *0.5 mm* *Curve length:* *230 mm*	
Corner radius: 2.5 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.5 kW		
Binding depth: 6.8 mm (average)		
Penetration depth: 7.6 mm (average)		
Notch size: 0.5 mm (average)		
Stiffener from profile number: 3		
Surface cracks except start/end: No		

Remarks:

Corner radius increased – The introduced error in the gap measurement is not compensated for

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	900	400
0.3	0,5	750	500
0.6	0,7	750	900
0.9	1,1	700	1650

Experiment ID: 73B2	Section A: *Binding depth:* *6.4 mm* *Penetration depth:* *8.6 mm* *Notch size:* *0.2 mm* *Curve length:* *230 mm* Section B: *Binding depth:* *6.0 mm* *Penetration depth:* *8.3 mm* *Notch size:* *0.4 mm* *Curve length:* *200 mm*	
Corner radius: 2.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.5 kW		
Binding depth: 6.2 mm (average)		
Penetration depth: 8.4 mm (average)		
Notch size: 0.3 mm (average)		
Stiffener from profile number: 3		
Surface cracks except start/end: No		

Remarks:

Corner radius increased – The introduced error in the gap measurement is not compensated for

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	900	400
0.3	0,5	750	500
0.6	0,7	750	900
0.9	1,1	700	1650

Experiment ID: 73B3	Section A: *Binding depth:* *5.7 mm* *Penetration depth:* *7.3 mm* *Notch size:* *0.4 mm* *Curve length:* *230 mm* Section B: *Binding depth:* *5.4 mm* *Penetration depth:* *7.2 mm* *Notch size:* *0.3 mm* *Curve length:* *200 mm*	
Corner radius: 1.5 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.45 kW		
Binding depth: 5.6 mm (average)		
Penetration depth: 7.2 mm (average)		
Notch size: 0.4 mm (average)		
Stiffener from profile number: 3		
Surface cracks except start/end: No		

Remarks:

Corner radius increased – The introduced error in the gap measurement is not compensated for

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	900	400
0.3	0,5	750	500
0.6	0,7	750	900
0.9	1,1	700	1650

Experiment ID: 73B4	Section A: *Binding depth:* *5.5 mm* *Penetration depth:* *8.1 mm* *Notch size:* *0.4 mm* *Curve length:* *230 mm* Section B: *Binding depth:* *5.1 mm* *Penetration depth:* *6.4 mm* *Notch size:* *0.2 mm* *Curve length:* *200 mm*	
Corner radius: 1.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.5 kW		
Binding depth: 5.3 mm (average)		
Penetration depth: 7.3 mm (average)		
Notch size: 0.3 mm (average)		
Stiffener from profile number: 3		
Surface cracks except start/end: No		
Remarks: Corner radius increased – The introduced error in the gap measurement is not compensated for		

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	900	400
0.3	0,5	750	500
0.6	0,7	750	900
0.9	1,1	700	1650

Experiment ID: 73C1	Section A: *Binding depth:* *4.1 mm* *Penetration depth:* *7.7mm* *Notch size:* *0.2 mm* *Curve length:* *230 mm* Section B: *Binding depth:* *4.3 mm* *Penetration depth:* *6.8 mm* *Notch size:* *0.2 mm* *Curve length:* *200 mm*	
Corner radius: 0.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.5 kW		
Binding depth: 4.2 mm (average)		
Penetration depth: 7.2 mm (average)		
Notch size: 0.2 mm (average)		
Stiffener from profile number: 3		
Surface cracks except start/end:		
Remarks: Displacement is decreased		

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	-0,1	900	400
0.3	0,2	750	500
0.6	0,4	750	900
0.9	0,8	700	1650

Experiment ID: 73C2	Section A: *Binding depth:* *5.2 mm* *Penetration depth:* *7.2 mm* *Notch size:* *0.3 mm* *Curve length:* *420 mm* Section B: *Binding depth:* *5.4 mm* *Penetration depth:* *7.5 mm* *Notch size:* *0.4 mm* *Curve length:* *390 mm*	
Corner radius: 0.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.6 kW		
Binding depth: 5.3 mm (average)		
Penetration depth: 7.3 mm (average)		
Notch size: 0.4 mm (average)		
Stiffener from profile number: 3		
Surface cracks except start/end: No		
Remarks: Displacement is increased		

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,5	675	600
0.3	0,8	563	750
0.6	1,0	563	1350
0.9	1,4	525	2475

Experiment ID: 73C3	Section A: *Binding depth:* *6.0 mm* *Penetration depth:* *8.0 mm* *Notch size:* *0.3 mm* *Curve length:* *200 mm* Section B: *Binding depth:* *5.1 mm* *Penetration depth:* *6.4 mm* *Notch size:* *0.1 mm* *Curve length:* *230 mm*	
Corner radius: 0.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.6 kW		
Binding depth: 5.6 mm (average)		
Penetration depth: 7.2 mm (average)		
Notch size: 0.2 mm (average)		
Stiffener from profile number: 3		
Surface cracks except start/end: No		
Remarks: Displacement is increased		

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,7	675	600
0.3	1,0	563	750
0.6	1,2	563	1350
0.9	1,5	525	2475

Experiment ID: 73E1	Section A: *Binding depth:* *4.9 mm* *Penetration depth:* *7.7 mm* *Notch size:* *0.3 mm* *Curve length:* *260 mm* Section B: *Binding depth:* *5.0 mm* *Penetration depth:* *7.6 mm* *Notch size:* *0.4 mm* *Curve length:* *230 mm*	
Corner radius: 0.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 6.1 kW		
Binding depth: 4.9 mm (average)		
Penetration depth: 7.7 mm (average)		
Notch size: 0.4 mm (average)		
Stiffener from profile number: 6		
Surface cracks except start/end: Yes		
Remarks: Laser Power is decreased		

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	900	400
0.3	0,5	750	500
0.6	0,7	750	900
0.9	1,1	700	1650

Experiment ID: 73E2	Section A: *Binding depth:* *5.5 mm* *Penetration depth:* *7.3 mm* *Notch size:* *0.4 mm* *Curve length:* *360 mm* Section B: *Binding depth:* *5.2 mm* *Penetration depth:* *7.8 mm* *Notch size:* *0.3 mm* *Curve length:* *330 mm*	
Corner radius: 0.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 6.9 kW		
Binding depth: 5.3 mm (average)		
Penetration depth: 7.5 mm (average)		
Notch size: 0.3 mm (average)		
Stiffener from profile number: 6		
Surface cracks except start/end: No		
Remarks: Laser Power is decreased		

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	900	400
0.3	0,5	750	500
0.6	0,7	750	900
0.9	1,1	700	1650

Experiment ID: 73E3	Section A: *Binding depth:* *6.0 mm* *Penetration depth:* *7.5 mm* *Notch size:* *0.3 mm* *Curve length:* *130 mm* Section B: *Binding depth:* *5.8 mm* *Penetration depth:* *8.4 mm* *Notch size:* *0.4 mm* *Curve length:* *100 mm*	
Corner radius: 0.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.9 kW		
Binding depth: 5.9 mm (average)		
Penetration depth: 7.9 mm (average)		
Notch size: 0.3 mm (average)		
Stiffener from profile number: 6		
Surface cracks except start/end: Yes		
Remarks: Laser Power is increased		

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	900	400
0.3	0,5	750	500
0.6	0,7	750	900
0.9	1,1	700	1650

Experiment ID: 73F1	Section A: *Binding depth:* *5.3 mm* *Penetration depth:* *8.3 mm* *Notch size:* *0.7 mm* *Curve length:* *230 mm* Section B: *Binding depth:* *5.7 mm* *Penetration depth:* *9.3 mm* *Notch size:* *0.3 mm* *Curve length:* *200 mm*	
Corner radius: 0.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.45 kW		
Binding depth: 5.5 mm (average)		
Penetration depth: 8.8 mm (average)		
Notch size: 0.5 mm (average)		
Stiffener from profile number: 3		
Surface cracks except start/end: No		
Remarks: Speed is decreased 50%		

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	450	400
0.3	0,5	375	500
0.6	0,7	375	900
0.9	1,1	350	1650

Experiment ID: 73F5	Section A: *Binding depth:* *4.3 mm* *Penetration depth:* *6.2 mm* *Notch size:* *0.1 mm* *Curve length:* *230 mm* Section B: *Binding depth:* *4.1 mm* *Penetration depth:* *6.0 mm* *Notch size:* *0.2 mm* *Curve length:* *200 mm*	
Corner radius: 0.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.45 kW		
Binding depth: 4.2 mm (average)		
Penetration depth: 6.1 mm (average)		
Notch size: 0.2 mm (average)		
Stiffener from profile number: 6		
Surface cracks except start/end: Yes		
Remarks: Speed is increased 50%		

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	1350	400
0.3	0,5	1125	500
0.6	0,7	1125	900
0.9	1,1	1050	1650

Experiment ID: 73F6	Section A: *Binding depth:* *4.6 mm* *Penetration depth:* *6.2 mm* *Notch size:* *0.1 mm* *Curve length:* *330 mm* Section B: *Binding depth:* *4.4 mm* *Penetration depth:* *5.5 mm* *Notch size:* *0.3 mm* *Curve length:* *300 mm*	
Corner radius: 0.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.5 kW		
Binding depth: 4.5 mm (average)		
Penetration depth: 5.9 mm (average)		
Notch size: 0.2 mm (average)		
Stiffener from profile number: 6		
Surface cracks except start/end: Yes		
Remarks: Speed is increased 25%		

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	1125	400
0.3	0,5	938	500
0.6	0,7	938	900
0.9	1,1	875	1650

Experiment ID: 73F7	Section A: *Binding depth:* *4.9 mm* *Penetration depth:* *8.5 mm* *Notch size:* *0.3 mm* *Curve length:* *230 mm* Section B: *Binding depth:* *5.2 mm* *Penetration depth:* *8.4 mm* *Notch size:* *0.2 mm* *Curve length:* *200 mm*	
Corner radius: 0.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.5 kW		
Binding depth: 5.1 mm (average)		
Penetration depth: 8.4 mm (average)		
Notch size: 0.3 mm (average)		
Stiffener from profile number: 3		
Surface cracks except start/end: No		
Remarks: Speed is decreased 25%		

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	675	400
0.3	0,5	563	500
0.6	0,7	563	900
0.9	1,1	525	1650

Experiment ID: 73G1	Section A: *Binding depth:* *4.3 mm* *Penetration depth:* *7.2 mm* *Notch size:* *0.3 mm* *Curve length:* *230 mm* Section B: *Binding depth:* *5.2 mm* *Penetration depth:* *7.9 mm* *Notch size:* *0.3 mm* *Curve length:* *200 mm*	
Corner radius: 0.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.45 kW		
Binding depth: 4.7 mm (average)		
Penetration depth: 7.6 mm (average)		
Notch size: 0.3 mm (average)		
Stiffener from profile number: 3		
Surface cracks except start/end: No		
Remarks: Wire feed rate set to zero		

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	900	0
0.3	0,5	750	0
0.6	0,7	750	0
0.9	1,1	700	0

Experiment ID: 73G2	Section A: *Binding depth:* *4.3 mm* *Penetration depth:* *7.3 mm* *Notch size:* *0.2 mm* *Curve length:* *230 mm* Section B: *Binding depth:* *5.1 mm* *Penetration depth:* *7.5 mm* *Notch size:* *0.3 mm* *Curve length:* *200 mm*	
Corner radius: 0.0 mm		
Work angle of laser beam: 19°		
Travel angle of laser beam: 0°		
Defocus length: -15 mm		
Work angle of wire and plasma gas: 45°		
Travel angle of wire and plasma gas: 23°		
Protection gas flow rate: 86 l/min		
Plasma control gas flow rate: 6 l/min		
Intentional size of gap: 0.0 mm	*Measured Gap:*	
Laser Power at weld scene: 7.45 kW		
Binding depth: 4.7 mm (average)		
Penetration depth: 7.4 mm (average)		
Notch size: 0.2 mm (average)		
Stiffener from profile number: 2		
Surface cracks except start/end: No		
Remarks: Wire feed rate is increased 50%		

Inverse Process Model	Welding Control variables		
Gap [mm]	Displace-ment [mm]	Speed [mm/min]	Wire Feed Rate [mm/min]
0.0	0,2	900	600
0.3	0,5	750	750
0.6	0,7	750	1350
0.9	1,1	700	2475

Appendix E

Introduction to Laser Welding

This appendix includes a report, which was written in the beginning of the present Ph.D. work. It gives a very basic introduction to laser technology and to the laser welding process.

Annex A

Manual for Sensor Based Laser Welding System

This annex is a manual for operators using the implemented laser welding system in the laser processing cell at Odense Steel Shipyard. The manual is written in Danish language.

Institut for Produktion
Aalborg Universitet

Department of Production
Aalborg University

Manual for Sensorstyret Lasersvejsning

Denne manual er skrevet i juli 1998, og beskriver fremgangsmåden ved gennemførelse af sensorstyret lasersvejsning på Lasercellen i hal B14 på Odense Stålskibsværft A/S.

Manualen beskæftiger sig kun med problemstillinger, der er specifikke for anvendelse af sensorbaseret styresystem til lasersvejsning, udviklet af Institut for Produktion på Aalborg Universitet i perioden 1. august 1995 til 1. juli 1998.

En forudsætning for anvendelse af denne manual er kendskab til anvendelse af MVS-sensor, samt til anvendelse af robotstyresystemet fra Power Automation. Derfor kan det være nødvendigt at anvende dokumentation for disse systemer sammen med denne manual.

Henrik John Andersen

Institut for Produktion

Aalborg Universitet

Fibigerstræde 16 · DK-9220 Aalborg Ø
Telf. 96 35 80 80 · Telefax 98 15 30 30

Fibigerstraede 16 · DK-9220 Aalborg · Denmark
Phone: +45 96 35 80 80 · Telefax: +45 98 15 30 30

Opsætning af systemet

For at kunne udføre sensorstyret svejsning, er det nødvendigt at systemets softwaremæssige forudsætninger er opfyldte. Disse forudsætninger består i:

1) Opsætning af PA-systemet.
2) Opsætning af Compile Cycles procedurer.
3) Opsætning af kalibreringsfiler til systemet.
4) Opsætning af Invers procesmodel.
5) Opsætning af Gpibkort og forbindelse til sensor-PC.

Opsætning af PA-systemet:
Opsætning af PA-systemet består i at lade indholdet i dedikerede opsætningsfiler fortælle at systemet anvender brugerfremstillede compilecyclesprocedurer, samt navnet på disse procedurer. Desuden skal der allokeres hukommelse til compilecycles enhederne.

Opsætningsfiler:
Det er muligt at sætte PA-controlleren op til sensorstyret lasersvejsning ved at anvende vedlagte diskette "Lasos disk1". Her kan de nødvendige filer sættes op pr. automatik, men hukommelse skal stadig allokeres som beskrevet i nedenstående afsnit om hukommelse.

Indhold af filer:
I det følgende beskrives indholdet i opsætningsfilerne, samt placeringen på PA-controllerens harddisk. En kopi af filerne findes på vedlagte diskette "Lasos disk 1".
fil: '**pa8000.ini**': Filen er placeret i biblioteket c:\p2. I denne fil defineres navnene på de compilecycles moduler, der anvendes. Dette sker for LASOS-systemet ved at finde det sted i filen, hvor der står :
'CZBFILE = ….. '
'CZSFILE = …...'
Såfremt der er semikolon foran disse skal dette fjernes, og så skal der iøvrigt stå:
'CZBFILE = BASIC'
'CZSFILE = SERVO4'
fil: '**cnc**': Filen er placeret i biblioteket c:\RMX386\CONFIG\USER. Filen indeholder en enkelt linie, og såfremt der står 'start.job' i denne linie, skal dette erstattes med 'czstart'.

Filerne '**pa8000.ini**' og '**cnc**' sættes op ved at stå i rodbiblioteket på 'Lasos disk1' og skrive '**weld**'. Filerne beskrives senere i manualen.

I PA-controlleren skal robotunderprogrammerne 222 og 333 være tilstede. Disse styrer henholdsvis start og stop af lasereffekt, tråd samt gas. Her sættes den absolutte lasereffekt og gasflow.

Hukommelse:
For at compilecycle enhederne 'basic' og 'servo4' kan foretage den fornødne kommunikation med hinanden, er det nødvendigt at der er afsat den fornødne hukommelse til dette. Denne hukommelse er sat op inde i menuen for PA-robotcontrolleren, under menuen: **INFO -> System commands -> Debugger** sættes adressen '**0D444**' til en værdi der er mindst 1 større end defineret fra Power Automation. Denne værdi bør være tilstrækkelig stor, såfremt den sættes til 90. Ændringer i robotcontrolleren kan dog forandre denne absolutte værdi.

Opsætning af Compile Cycles procedurer:
fil: '**basic**': filen er en kompileret compilecycle enhed, og skal være placeret i bibilioteket 'c:\p2'.

fil: '**servo4**': filen er en kompileret compilecycle enhed, og skal være placeret i bibilioteket 'c:\p2'.

Såfremt filerne ikke findes på PA-controlleren, må '**basic**' og '**servo4**' kopieres manuelt fra a:\p2 ('lasos disk 1') til c:\p2 på PA-controlleren.

Opsætning af kalibreringsfiler til systemet:
fil: '**scanpos.ini**': Filen er en kalibreringsfil til kalibrering af MVS-sensorens position i forhold til laserstrålens fokuspunkt (og den ønskede defokusering). Filen er placeret i biblioteket c:\p2. filens struktur, og fremgangsmåden for kalibreringen er beskrevet senere i denne manual.

fil: '**Start.ipm**': Filen indeholder definitionen af den inverse procesmodel, der anvendes til styring af svejseprocessen. Desuden er den ønskede svejselængde defineret i denne fil. Filen er placeret i biblioteket 'c:\p2'. Filens struktur og indhold er beskrevet senere i denne manual.

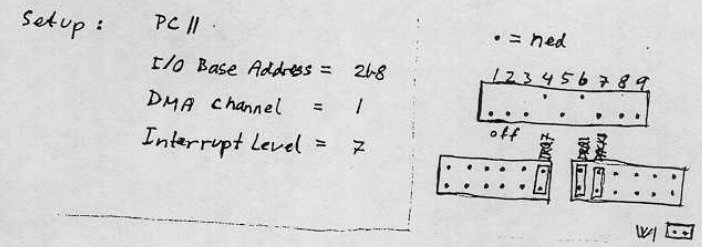
fil: '**kalib.ipm**': Filen svarer til '**start.ipm**', men anvendes udelukkende til kalibrering af det sensorbaserede styresystems positioneringsnøjagtighed (se dette afsnit senere). Brugeren kommer aldrig direkte i kontakt med kalib.ipm, men det er vigtigt at den er tilstede i biblioteket *c:\p2'. Indholdet af filen beskrives under afsnittet "kalibrering af det sensorbaserede styresystems positioneringsnøjagtighed".

Filerne '**scanpos.ini**', '**start.ipm**' og '**kalib.ipm**' overføres til PA-controllern ved at stå i rodbiblioteket på 'Lasos disk1' og skrive '**setup**'. Filerne beskrives detaljeret senere i manualen.

Opsætning af GPIB-kort og forbindelse til sensor-PC:
Styresystemet anvender GPIB til kommunikation mellem sensor-PC og PA-controller.

Software: Softwaren til Gpib-kortet i PA-controlleren består af et softwaremodul i compilecycles-enheden basic. Dette modul hedder "gpibdrv.c" og "gpibdrv.h". Disse moduler er tilrettet til Lasos-applikationen med basis i den gennerelle åbne Gpibsoftware "esp488" fra National Instruments. Denne software ligger på vedlagte diskette "Lasos disk2". Den tilrettede software til applikationen kan findes sammen med de øvrige compilecycles procedurer på vedlagte diskette "LASOS disk1".
På Sensor-PC'en kører GPIB-som beskrevet i manualerne for GPIB-interfacet. (se manualer fra National Instruments).

Hardware: For at softwaremodulet gpibdrv kan håndtere kommunikationen, må GPIB-kortet være konfigureret som vist her:



MVS-sensor:

MVS-sensoren tændes på den sorte styringsbox, ved at trykke på START. Knappen MANUEL må ikke være aktiveret. Husk at nøglen skal være drejet.

Foruden Selve sensoren, skal Sensor pc'en være tændt. Når denne tændes starter sensorens styringssoftware automatisk. Ellers kan man gå ind i biblitoteket C:\jms>. Ved at skrive "j" efterfulgt af "ENTER", vil styringssoftwaren starte, og sensorprogrammets skærmside komme frem.

Inden gennemførelse af svejsning skal det kontrolleres at MVS-sensoren er klar. Dette. Hvis dette er tilfældet vil feltet "Gpib" i sensor-pc'ens menubjælke er markeret. Desuden må sensoren ikke være aktiv, d.v.s. den må ikke være igang med at måle. Dette kontrolleres ved at "sensorstatus" under det sorte felt ikke har nogen værdi.

PA-Controller:

Overordnet beskrivelse

Udførelse af sensorbaseret svejsning foregår ved at der laves et CNC-programm, hvor robotten fører MVS-sensoren til en position, hvor den kan se svejsefugens startpunkt. Her startes sensoren automatisk og svejseudstyret flyttes automatisk til den position, hvor laserstrålens fokuspunkt vil ramme startpunktet for svejsningen.

Når svejseudstyret er flyttet til startpunktet stopper eksekvering af CNC-programmet. For at igangsætte selve svejsningen skal operatøren starte eksekvering af CNC-programmet igen.

Når CNC-programmet igen startes, kaldes et underprogram (program 222) der styrer opstart af laserkilden og og eventuelle forsinkelser inden bevægelsen starter. Resten af svejsningen udføres automatisk af det sensorbaserede styresystem, indtil den specificerede svejselængde er nået, hvilket beskrives senere.

Når den specificerede svejselængde er udført, kalder styresystemet underprogrammet 333. Dette styrer stopproceduren uden bevægelse, hvorunder laseren slukkes, tråden trækkes tilbage o.s.v..

Efter slukning af laseren eksekveres resten af hovedprogrammet. Her flyttes robotten evt. væk fra svejsesømmen.

Procedurebeskrivelse for udførelse af lasersvejsning

Strukturen for hovedprogrammet for sensorbaseret lasersvejsning er beskrevet i det følgende. Denne struktur sikrer at svejsefugens startpunkt er kendet, samt at hastigheden hvormed fokuspunktet flyttes til startpunktet er sat.

Den del af hovedprogrammet, der anvendes af det sensorbaserede styresystem er beskrevet i det følgende. Hovedprogrammet kan have et vilkårligt udseende både før og efter denne del af programmet.

Strukturen af den del af hovedprogrammet der anvendes af det sensorbaserede styresystem, er herunder vist og forklaret ved et eksempel.

Eksempel på programstruktur:

N210 X9334 Y1280 Z137 B-70 C87 G400

N220 G1 F2000

N230 G1 F2000

N240 G500

N250 G04

N260 Q333

Forklaring af programstruktur:

Linienumre, positioner og hastigheder er valgt tilfældigt. Dog bør hastigheden i linierne 220 og 230 aldrig overstige 2000mm/min.

Linie 210: Denne linie beskriver robottens position når MVS-sensoren er placeret ud for svejsesømmens startpunkt. Dette er markeret ved koden G400. Det er vigtigt at MVS-sensoren kan se svejsefugen i denne position, samt at alle aksernes positioner (X,Y,Z,B og C) er specificeret i denne position. Man skal være opmærksom på, at svejsevinklen skal være den rette når robotten stilles i denne position, da denne ikke bliver reguleret af det sensorbaserede styresystem.

Linie 220 og 230: Disse linier skal være éns og de specificerer at der skal køres i en ret linie med hastigheden 2000mm/minut (gerne mindre) til positionen hvor laserstrålens fokuspunkt rammer svejsesømmens startpunkt.

Linie 240: Denne kode fortæller at hovedprogrammet ikke indeholder flere oplysninger vedrørende svejseopgaven.

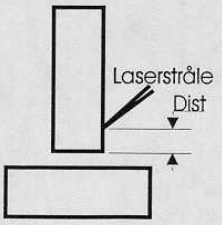
Linie 250: Denne linie er en såkaldt "dummy-linie", som er krævet i forbindelse med afslutningen af det sensorbaserede styresystem.

Linie 260: Denne linie kalder et underprogram, der afslutter svejsningen efter fremføringsbevægelsen er afsluttet.

Invers procesmodel

For at lade sensorsystemet styre svejseprocessen, er det nødvendigt at definere svejse- og trådhastigheder, samt placering af fokuspunkt i forhold til svejsefugen. Dette er gjort i en invers procesmodel. Desuden skal svejselængden for hver enkelt svejseopgave defineres. Både den inverse procesmodel og svejselængden er defineret i filen "c:\p2\start.ipm". Et eksempel på denne fil er vedhæftet som bilag.

Til en given gabsstørrelse hører en 'dist', en wire speed, og en welding speed. Dist er defineret som afstanden mellem kanten på det lodrette emne og laserstrålens angrebspunkt på dette. Dist er positivt målt i retningenvæk fra det vandrette emne, og er angivet i mm.



Filen "start.ipm" har en struktur som vist herunder:

```
start.ipm
Thickness of Body: 10mm
Thickness of flange: 10mm
Power at workplace: 8.9kW
Incident Beam Angle: 20
Angle between wire and welding direction: 45
Wire Thickness

Gap     Defocus Dist WireSp. WeldingSp.
0.0     0       0.2  400     900
0.3     0       0.3  500     750
0.6     0       0.4  650     675
0.9     0       0.5  1400    600
1.2     0       0.6  2400    550
2.0     0      -2.0  850     650

WeldLength:   50 mm
end
```

Det øverste af filen er et hoved, som angiver filens navn samt de forudsætninger hvorunder procesmodellen er gyldig. Herunder er angivet en tabel, hvori værdierne for styrevariablene er defineret. Til sidst er længden af den pågældende svejsefuge defineret. Det er vigtigt at man ikke ændrer i teksten, men kun i værdierne i denne fil.

Det kan ses at der i filen gives mulighed for at definere en bestemt defokuslængde. Denne er dog ikke implementeret i styresystemet endnu, hvorfor den altid skal være sat til værdien 0.

Som det fremgår af filen, er der 6 rækker i den inverse procesmodel. Dette skal der altid være og værdiernde for gab skal være forskellige i de 6 rækker. Den første skal altid være 0 og den sidste 2,0.

Hæftninger:
Når sensoren kører over hæfninger køres efter de værdier der er defineret for gab = 2.0mm (se filen start.ipm). Her gælder særlige forhold for parameteren DIST. Den ønskede værdi af DIST skal nemlig i dette tilfælde fratrækkes værdien 2.0. Vil man f.eks. lade angrebspunktet være 0.5mm over den lodrette plades kant, skal man sætte DIST til 1.5mm.

Et andet forhold omkring hæftninger er, at der ikke må være noget gab mellem emnerne under en hæftning. Dette skyldes at sensoren ikke er i stand til at måle dette gab, og ej heller kan trække pladerne sammen under svejseprocessen (ingen lynlåseffekt).

Kalibrering af det sensorbaserede styresystems positioneringsnøjagtighed

Kalibrering af laserstrålens position vinkelret på svejsefugen:

Kalibrering af fokuspunktets position vinkelret på svejsefugen foretages ved, at lade det sensorbaserede styresystem føre robotten langs en svejsefuge uden at

tænde laseren. Et sted på svejsefugen stoppes eksekveringen og der afgives et skud med laseren, hvorefter fokuspunktets position i forhold til svejsefugen kan vurderes og evt. ændres jf. nedenstående.

Fremgangsmåde:

1) Der opstilles en kantsøm, hvor kanterne på det lodrette emne er helt skarpe (fræste kanter).
2) Der laves et robotprogram med en svejsedel, som vist under afsnittet ' Procedurebeskrivelse for udførelse af lasersvejsning'. Dog skal værdien G400 i linie 210 erstattes af G401.
3) Programmet eksekveres under tørløb, dvs. knappen 'laser on/off' sættes til laser off.
4) Når laserstrålens position befinder sig et sted mellem start- og slutpunkt for svejsningen stoppes programeksekveringen, ved at trykke på 'Reset CNC'.
5) Der laves et enkelt skud med laseren i denne position.
6) Robotten flyttes et par centimeter positivt i Z-retningen og der laves endnu et enkelt skud.
7) Robotten køres væk, således at det er muligt at vurdere placeringen af de to skud på den lodrette plade. Positionen af det første skud skal være placeret således, at centrum af skuddet ligger præcist på kanten af den lodrette plade. Det halve af dette skud skal således være placeret på den lodrette plade. Det andet skud tjener det formål at vurdere størrelsen af et helt skud. Det er derfor vigtigt at robotten KUN er flyttet i Z-retningen inden afgivelse af dette, da fokuseringen og dermed størrelsen af skuddets omkreds ellers vil være ændret.
8) Såfremt det første skud ikke er placeret således at skudcirklens centrum ligger præcist på kanten af det lodrette emne, kan det konstateres at laserstrålen rammer forkert på emnet. Dennes position flyttes ved hjælp af værdien "delta_Ly" i kalibreringsfilen 'c:\p2\scanpos.ini'. Værdien ændres i positiv retning for at flytte laserstrålens position længere ned ad den lodrette plade. delta_Ly er angivet i mm.

punkt 3) til 8) køres igennem indtil det første skud rammer præcist med skuddets centrum på kanten af det lodrette emne. Efter endt kalibrering skal værdien G401 i hovedprogrammet igen erstattes af G400 for at køre normal svejsning.

Filen '**kalib.ipm**': Denne fil anvendes af det sensorbaserede styresystem, når koden G401 er angivet i hovedprogrammet. Dette er tilfældet under kalibrering af laserstrålens position vinkelret på svejsefugen. Indholdet af filen svarer til indholdet i '**start.ipm**', som er beskrevet i forrige afsnit. Dog er der den vigtige undtagelse at værdien DIST ***ALTID*** er nul, uanset gabets størrelse. Dette gør at et laserskud altid rammer med centrum på den lodrette plades kant.

Kalibrering af laserstrålens fokusering:

Det skal med mellemrum kontrolleres at Laserstrålen har den rigtige fokusering i forhold til emnernes overflader. For at kontrollere dette må man i forvejen

kende afstanden mellem gasdysen og laserstrålens fokuspunkt. Desuden skal man kende den ønskede størrelse af defokusering, hvorved man kan beregne den ønskede afstand fra gasdysen til svejsefugens overflade.

Fremgangsmåde:

1) Der opsættes et emne til svejsning af kantsøm.
2) Der laves et robotprogram med en svejsedel, som vist under afsnittet ' Procedurebeskrivelse for udførelse af lasersvejsning'.
3) Programmet eksekveres under tørløb, dvs. knappen 'laser on/off' sættes til laser off.
4) Når laserstrålens position befinder sig et sted mellem start- og slutpunkt for svejsningen stoppes programeksekveringen, ved at trykke på 'Reset CNC'.
5) Såfremt afstanden mellem gasdysen og svejsefugens overflade ikke svarer til den ønskede afstand ændres værdien 'L_sensor' i filen "c:\p2\scanpos.ini". Hvis denne værdi gøres større, flytter man fokuspunktet ud fra svejsefugens overflade. L_sensor er angivet i mm.

Punkterne 3) til 5) gentages indtil afstanden mellem gasdyse og svejsefugens overflade svarer til den ønskede værdi.

Kalibrering af afstanden mellem MVS-sensor og fokuspunkt.

Med mellemrum bør det kontrolleres om afstanden mellem MVS-sensoren og laserstrålens fokuspunkt er kalibreret korrekt. Denne afstand kan ændre sig ved oplinering af strålen eller ved flytning af sensoren. Desuden kan der forkomme små ændringer ved af- og på montering af sensoren samt ved kollisioner.

Fremgangsmåde:

1) Der opstilles en kantsøm, hvor kanterne på det lodrette emne er helt skarpe (fræste kanter).
2) Der laves et robotprogram med en svejsedel, som vist under afsnittet 'Procedurebeskrivelse for udførelse af lasersvejsning'.
3) Programmet eksekveres under tørløb, dvs. knappen 'laser on/off' sættes til laser off.
4) Når laserstrålens position befinder sig et sted mellem start- og slutpunkt for svejsningen stoppes programeksekveringen, ved at trykke på 'Reset CNC'.
5) Der laves et enkelt skud med laseren i denne position.
6) MVS-sensoren tændes manuelt, således at den står og foretager målinger.
7) Der føres en lineal med en vis tykkelse ind over svejsefugen, således at den skubbes fra mærket af laserskuddet og frem mod MVS-sensoren. Når linealen lige netop bliver til syne på MVS-sensorens målinger (de to hvide streger på sensorens monitor), aflæses afstanden fra skudmærket i svejsefugen til den ende af linealen der vender mod sensoren.
8) Den aflæste afstand skal svare til værdien 'L_lookahead' i filen "c:\p2\scanpos.ini". Hvis nødvendigt ændres denne værdi.

Annex B

Material Specifications

This annex is a copy of the material Specifications for Steel plates used for the procedure tests of the approved inverse process model. The same type of material was used for the experiments presented in chapter 7.

The specification from the Danish steel mill ‘Det Danske Stålvalseværk’ is on the metal sheets used for the deck plates, or for workpiece 2 in the experiments.

The specification from fundia is on the steel profiles used as stiffeners or as workpiece 1 in the experiments presented in chapter 7.

26/01 '01 13:29 FAX 4563972318 OSS PTA 004

10

Det Danske Stålvalseværk A/S

DK-3300 Frederiksværk - Telefon 47 77 03 33 - Telefax 42 12 46 66 - Telex 40191

BESCHEINIGUNG/CERTIFICATE

Bleche / Plates

Seite/Page: 1.1 Nr./No.: 98836

Type: AB (2)
AB

Ihrer Auftrag/Your order: 1-43301/164/LASER

Unser Auftrag/Our order: 24530

Datum/Date: 20.01.1998

9004 AA24 00

ODENSE STAALSKIBSVÆRFT A/S
POSTBOX 176
5100 ODENSE

Lieferstelle/Delivery address:
LOSSESTED:LINDØVÆRFTET,ODENSE VÆF

NYBYG: 164

Lieferbedingung / Specification: ABS A (LASERWELD)

Lieferung/Delivery: M/S CETUS

Toleranz/Tolerance: AB

	Pos.	Zeichen/Marking	Abmessungen/Dimensions				Stk./Pcs.	Gewicht/Weight	Schmelze/Heat	Slab	Walznr./Millno.	
1	3581	026118-752	12500	1800	10,0		1	1800	44378	A1	5709K 0	–
2	3581	026118-752	12500	1800	10,0		1	1800	44378	C3	5716K 0	–
3	3584	026121-752	13140	1800	10,0		1	1892	44174	E1	3706K 0	–
4	3586	026123-752	14030	1800	10,0		1	2020	44378	A3	5714K 0	–
5	3588	026125-752	15560	1800	12,0		1	2689	44378	B5	5089K 0	–
6	3588	026125-752	15560	1800	12,0		1	2689	44378	B1	5090K 0	–
7	3588	026125-752	15560	1800	12,0		1	2689	44378	B2	5711K 0	–
8	3588	026125-752	15560	1800	12,0		1	2689	44378	C5	5712K 0	–
9	3588	026125-752	15560	1800	12,0		1	2689	44378	B3	5713K 0	–
10	3588	026125-752	15560	1800	12,0		1	2689	44378	A5	5715K 0	–
11	3588	026125-752	15560	1800	12,0		1	2689	44378	C4	5717K 0	–
			Gewicht durch nominelle Abmessungen berechnet Calculated weight based on nominal dimensions				11	26335				

	C	Mn	Si	P	S	Cr	Cu	Ni	Mo	Sn	Al	Nb	Ti	V	B	N	Ceq = Carbon-Equivalent (IIW - formula)
1	11	69	25	12	3	6	25	9	1		32	0	0	3		65	26
2	11	69	25	12	3	6	25	9	1		32	0	0	3		65	26
3	9	70	23	12	5	8	26	7	1		32	0	0	4		61	25
4	11	69	25	12	3	6	25	9	1		32	0	0	3		65	26
5	11	69	25	12	3	6	25	9	1		32	0	0	3		65	26
6	11	69	25	12	3	6	25	9	1		32	0	0	3		65	26
7	11	69	25	12	3	6	25	9	1		32	0	0	3		65	26
8	11	69	25	12	3	6	25	9	1		32	0	0	3		65	26
9	11	69	25	12	3	6	25	9	1		32	0	0	3		65	26
10	11	69	25	12	3	6	25	9	1		32	0	0	3		65	26
11	11	69	25	12	3	6	25	9	1		32	0	0	3		65	26
		⅟₁₀₀		⅟₁₀₀₀			⅟₁₀₀				⅟₁₀₀₀					⅟₁₀.₀₀₀	⅟₁₀₀

Lieferzustand/Condition of Delivery
N = Normalgeglüht/Normalized 900°C
Dicke/Thickness :
– 25 mm min. 3 Minuten/Minutes
25,1 – mm min. 5 Minuten/Minuten
– = Walzzustand/as rolled
Maßkontrolle, Besichtigung / Dimension check, inspection
Ohne Beanstandung/Without complaint

Kennzeichnung/Marking
VAB

Stahlsorte/Quality
AB A

Sachverständigen/Quality inspector:

Schmelz nr. / Heat no.
1-89.999 Electro-Ofen/El-arc-furnace
Im Pfanne raffiniert/Laddle refined
90.000-99.999 Konverter-verfahren/Oxygen converter

	Zugversuch quer/Tensile test transverse						Kerbschlagbiegeversuch/Impact test												Warmzugversuch/Hot tensile test	
	Re	Rmt	A	Re	Rmt	A	1	2	3	Mittel/Average	Temp	1	2	3	Mittel/Average	Temp	*	$R_{0,2}$	Temp
1	315	440	30																
2	315	440	30																
3	309	434	29																
4	315	440	30																
5	315	440	30																
6	315	440	30																
7	315	440	30						—										
8	315	440	30																
9	315	440	30																
10	315	440	30																
11	315	440	30																
	N/mm²		⅟₁	N/mm²		⅟₁	Joule				°C	Joule				°C		N/mm²	°C

AMERICAN BUREAU OF SHIPPING ABS GOTHENBURG

Dehnung/Elongation: A200

We hereby certify that this Material described herein has been made to the applicable specification by the process stated above and tested in accordance with the requirements of ABS Rules with satisfactory result.
The Material in this certificate has been manufactured and tested in accordance with ABS Quality Assurance Program / Certificate 95-QA1125-X.
This certificate is only valid if endorsed and stamped by a Surveyor to the Bureau.

* Probelage und Zustand
Testpiece location and condition

Probebreite Width of testpc mm	Lage/location
1 10.0	Oberfläche/Surface
2 10.0	Mitte/Center
3 10.0	1/3t von Oberfl./from surface
4 10.0	1/4t von Oberfl./from surface
5 5.0	
7 7.5	
9 Blechdicke/Plate thickness	

Zustand/Condition

Faltversuch/Bend Test —

Wir bestätigen, dass die Lieferung den Anforderungen der obengenannten Lieferbedingungen entspricht.
We hereby certify, that the materiel has been made a tested in accordance with the mentioned specification

21 JAN. 1998

Z. Greisen
Chief Metallurgist
Det Danske Stålvalseværk A/

26/01 '01 13:29 FAX 4563972318 OSS PTA 005

fundia

INTYG
CERTIFICATE
ZEUGNIS

368/98

10

CERTIFICATE 3.1C EN 10204

98 SM 18787

Beställare/Customer/Besteller
ODENSE STAALSKIBBSVAERFT A/S

Stålsort/Steel grade/Stahlsorte	Beställarens order nr/Customer's order No./Auftrags-Nr. des Bestellers	Vår order/Our order No./Unsere Auftrags-Nr.
GRADE AL FLAT BARS	165-1-40137	3007/19173

HÅLLFASTHETSDATA MECHANICAL PROPERTIES FESTIGKEITSEIGENSCHAFTEN

Charge Cast Schmelze	Dimension Dimension Abmessung	Sträckgräns Yield stress Streckgrenze	Brottgräns Tensile strength Zugfestigkeit	Förlängning Elongation Dehnung		Hårdhet Hardness Härte	Slagseghet Impact value Kerbschlagzähigkeit	
		R_{eH}	R_m	A_5		H B	ISO KV	
Nr No.	mm	N/mm²	N/mm²	%	%		°C	Joule
4-3658	150,00 10,00 4690 Kg	319	463	35				

We hereby certify that the material described has been approved by and in accordance with the Rules of American Bureau of Shipping The material in question has been tested in the presence of the Society's representative with satisfactory results.

CHARGEANALYS % CAST ANALYSIS % SCHMELZANALYSE %

Charge nr Cast No. Schmelze Nr	C	Si	Mn	P	S	AL
4-3658	,09	,31	1,06	,011	,004	,020

Each bundle marked with attached metal tag with charge and dim.

AMERICAN BUREAU OF SHIPPING ABS SURVEYOR

SURVEYOR

SMEDJEBACKEN 98-02-20 KURT EKHOLM Kval.avd/QA

97-10 ROGERIGS FALUN 023-18150 11451 MO

Annex C

Approved Procedure Test

This Annex includes a copy of the approved procedure test for utilising the inverse process model for welding flange joints with a gap of maximum 0.5 mm. The procedure test was approved by the classification societies American Bureau of Shipping (ABS) and Lloyds Register of Shipping. The approval is signed on page 4 in this annex.

26/01/2001 13:22 4563972318 OSS PTA 001
6/01 '01 13:29 FAX 4563972318

ODENSE LINDØ

ODENSE STAALSKIBSVÆRFT A/S
ODENSE STEEL SHIPYARD LTD.

SVEJSELABORATORIET
WELDING LABORATORY

SAG NR. CASE NO.	DATO DATE	SIDE PAGE
WELD-LAB. NO: 97.30	980303.	1.

EMNE
SUBJECT

LASER-WELDING.
Deposited Metal Test for Laser-Welding
of T- connections in weld.pos.: PB, with
gap ~ 0,5 mm , and with ESABs wire OK 12.51,
all in acc. to WPS no: 751.002

OMFANG
EXTENT

SEE BELOW.

GRUNDLAG
BASIS

THE CLASSIFICATION SOCTIETIES REQUREMENTS FOR
APPROVAL OF CO_2 LASER WELDING PROCEDURES.

RESULTAT
RESULT

1. VISUALLY EXAMINATION BY ABS' SURVEYOR MR. J. ABSCHNEIDER.
2. MPI-, AND ULS. TESTS, SEE FORCE REPORT, PAGE 7 + 8.
3. X- RAY TEST, SEE FORCE REPORT, PAGE 9.
4. BENDING TEST, SEE OS-REPORT, PAGE 10.
5. IMPACT TEST, SEE OS-REPORT, PAGE 10.
6. BREAKING TEST, SEE OS-REPORT, PAGE 10.
7. HARDNESS TEST, SEE OS-REPORT, PAGE 11 + 12.
8. MACRO EXAMINATIONS, SEE OS-REPORT, PAGE 13.

028 A - 02-78

	UDFØRT/REPORT	
	AF/BY MH.	DATO/DATE 980406.

26/01 '01 13:29 FAX 4563972318 OSS PIA 002

D

WELDING PROCEDURE SPECIFICATION FOR FUSION WELDING / *PROCEDURESPECIFIKATION FOR SMELTESVEJSNING* **WPS**	PROCEDURE NO. 751.002 / *PROCEDURE NR.*	REVISION *RETTET*	DATE 980406

CONTRACTOR / *VÆRKSTED*	ODENSE STEEL SHIPYARD LTD. / ODENSE STAALSKIBSVÆRFT A/S	PRELIMINARY *FORELØBIG* ☐	FINAL *ENDELIG* ☒
WELDING METHOD / *SVEJSEMETODE*	LASER WELDING / LASER-SVEJSNING	WORKSHOP ERECTION *VÆRKSTED* ☐ *MONTAGE* ☐	WELD. POS. 2 F / *SVEJSEST.* PB.ON/

BASE MATERIAL *GRUNDMATERIALE*	☐ CHARGE	☒ SPECIFIC MATERIAL, SEE *ENKELT MATERIALE*, SE	BASE MATERIAL GROUP *MATERIALEGRUPPE OMFATTENDE* ☒ GRADE-TYPE
DESIGNATION *BETEGNELSE*	GUIDESLINES FOR APPROVAL OF CO_2-LASER WELDING § 5.		LR.1.L
HEAT NO. *CHARGE NR.*	RETNINGSLINIER FOR GODKENDELSE AF CO_2-SVEJSNING § 5.		

MATERIAL THICKNESS	PIPEDIAMETER	QUALITY DEMANDS ISO/EN 13.919
8 MM TO *TIL* 10 MM	MM TO *TIL* MM	CLASS *KARAKTER* C C

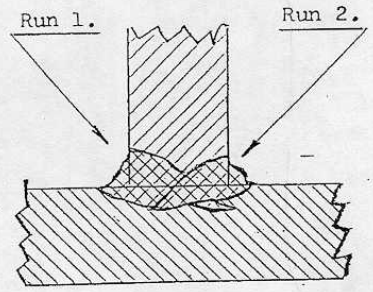
MATERIAL REQUIREMENTS *MATERIALEMÆSSIGE KRAV*	WELD METAL *SVEJSEMETAL*	HAZ *VARMEPÅVIR. ZONER*
TENSILE TEST, STRENGTH *TRÆKPRØVER, BRUDSTYRKE* N/MM²	400-560	
TENSILE TEST, ELONGATION *STRÆKPRØVER, BRUDFORLÆNGELSE* %	22% in 50 mm 22% på 50 mm	
ADDITIONAL REQUIREMENTS, IMPACT *TILLÆGSKRAV, SLAGSEJHEDSKRAV* J	47 by + 20 °C. 47 ved	None Ingen
ADDITIONAL REQUIREMENTS, HARDNESS *TILLÆGSKRAV, HÅRDHEDSMÅLING* HV10	Max. 380	Max. 380
ADDITIONAL REQUIREMENTS *TILLÆGSKRAV* COD	See Guideslines Se retningslinier	

GROOVE PREPARATION *FUGETILDANNELSE*	CLEANING: INITIAL BURN.+ BRUSH. PROFILE: MILLED *AFRENSNING: FØR SV.* BRÆN.+BØRST. PROFIL : FRÆSET.

GROOVE, TYPE AND DEFINITION *FUGEPROFIL*

BEAD SEQUENCE BY NUMBERING *SØMOPBYGNING, SVEJSERÆKKEFØLGE*

Scale: 2:1



TACKING, CLAMPING, BACKING *OPHÆFTNING, OPSPÆNDING, BAGSKINNE*

1 tacking pr. metre

1 hæftning pr. m.

Gap
Gab ~ 0,5 mm.

FILLERMETAL *TILSATSMATR.*	BRAND NAME *HANDELSBETEGNELSE*	SUPPLIER *LEVERANDØR*	CLASSIFICATION *KLASSIFIKATION*
A	OK 12.51	ESAB, Copenhagen	AWS: A5.18: ER 705-6
B			
C			

26/01 '01 13:29 FAX 4563972318 OSS PTA 003

PREHEAT TEMPERATURE / FORVARMETEMPERATUR
FROM None
OVER mm °C

PROCEDURE NR.: 751.002

PAGE 2 / SIDE 2

MIN. WORKSHOP TEMP. / MIN. VÆRKSTEDSTEMP. +5 °C

MAX. INTERPASS TEMP. / HØJESTE ARBEJDSTEMP. None / INgen

BACK GOUGING / OPFUGNING None / Ingen

POST HEAT TREATMENT / VARMEBEHANDLING None / Ingen

DRYING ETC. OF FILLER MATERIALS / TØRRING M.V. AF TILSATSMATERIALE According to manufacturers recommendations. / I.h.t. leverandørers henvisninger.

WELDING DATA BEAD NO. / STRENG NR.	FILLER MATR. / TILSATS-MATERIALE	WIRE DIEMSION mm / TRÅD DIMENSION mm	LASER POWER KW / LASER-STYRKE KW	WIRE SPEED cm/min. / TRÅD HAST. cm/min.	Focal dia. m m. / Focus dia. m m.	TRAVEL SPEED CM/MIN. / FREMFØ-RINGSHAS-TIGHED CM/MIN.	Focal LENGTH MM / Focus LÆNGDE MM	HEAT INPUT MJ/M / VARME-TILFØRSEL MJ/M	WEAVIN / PENDLIN
1	A	1,0	8,0±5%	70	0,6	72.5	400	~0,66	None
2	A	1,0	8,0±5%	70	0.6	72.5	400	~0,66	None

Controlled by adaptive control:
Welding speed. wire speed
Pos. of focal rel. to workpiece

ADDITIONAL FILLERMATERIAL / EKSTRA TILSATSMATERIALE None / Ingen

POSITIONING OF FILLER MATERIAL / PLACERING AF TILSATSTRÅDE None / Ingen

SHIELDING GAS / BESKYTTELSESGAS He — COMPOSITION / ART 100% He — CONSUMPTION / FORBRUG 10@ l/min. — NOZZLE DIAMETER / DYSEDIAMETER

PLASMAGAS / PLASMAGAS He — COMPOSITION / ART 100% — CONSUMPTION / FORBRUG 10 l/min.

PULSE FREQUENSE ETC. / PULSFREKVENS M.V. None / Ingen

REMARKS / BEMÆRKNINGER SEE / SE

GUIDESLINES FOR APPROVAL OF CO_2-LASER WELDING

RETNINGSLINIER FOR GODKENDELSE AF CO_2-SVEJSNING

WE HEREBY CERTIFY THAT THIS [ ] PRELIMINARY

[X] FINAL WPS IS IN ACCORDANCE WITH OUR PRODUCT

FACILITIES AND IS EXPECTED TO RESULT IN WELDI

SEAMS WITH MENTIONED QUALITIES

ODENSE STEEL SHIPYARD LTD.

DATE/SIGN. 980406. Møller Hansen

ENCLOSED CERTIFICATES:

- STEEL CERTIFICATE NO.
- RECORDED WELDING DATA NO. Weld.Lab.
- RECORDED DT. NO. Report No.:
- RECORDED NDT NO. 97.30

APPROVED BY CLASSIFICATION SOCIETY

ABS

LLOYD'S REGISTER OF SHIPPING 15/4/98 ODENSE

:6/01 '01 13:29 FAX 4563972318 OSS PTA 006

WELDING PROCEDURE

Page:
Report No.: 97.3o

PROCEDURE NR.: 751.oo2	POSITION: PB.
MATERIAL: fun. 98 SM 18787 CERTIFICATES DDS. 98836	DIMENSION: T- 15o·15o·1o·1285 mm.

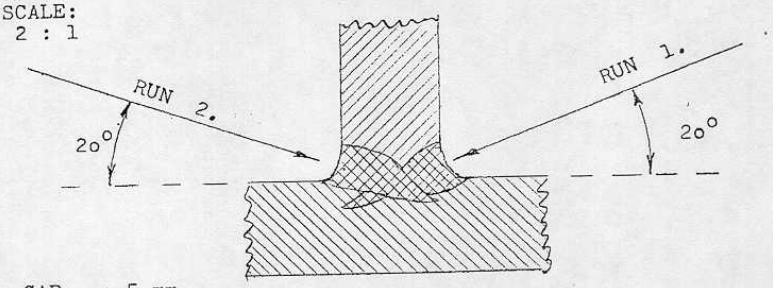
WELDINGMASHINE TYPE		ELECTRODES		CLASS
1.PAS: REST :	WB 12ooo.	1.PAS: REST :	ESAB OK 12.51- 1,omm	AWS ER7oS

NAME OF OPERATOR 1.	Mads Elvang	3732
NAME OF OPERATOR 2.	Michael Lodffersen	o952

REMARKS:

Gas	Gastype	Flow 1/min±10
Plasma control	Helium	10
Shielding	Helium	100

PROFILE:
SCALE:
2 : 1



GAP o,5 mm.

PAS NO:	WIRE DIM Ø	LASER KW.	WELD-SPEED mm/min max min.	WIRE SPEED mm/min max min	FOCAL dia.	"f" mm.	HEAT-INPUT MJ/m.
1	1,o	8,o	725	7oo	o,6	4oo	o,66
2	1.o	8,o	725	7oo	o,6	4oo	o,66

CONTROLLED BY ADAPTIVE CONTROL:

WELDING SPEED
WIRE SPEED.
POSITION OF FOCAL RELATIVE TO WORKPIECE.

Date: 98o328	Sign.: ME / MH.		Cont. page:

26/01 '01 13:29 FAX 4563972318 OSS PTA 007

Markrapport
Magnetpulverprøvning

FORCE INSTITUTTET

Division for Inspektion

DANAK Reg. nr.	Bilag nr.	Sag Id./Sag nr.	Rapport nr.	Side af
		S 22591.	M 64.	1/1.

Rekvisitions nr.	Kontrolperiode/Dato	Tekniker (Init.)	Certifikat nr.	Assistent (Init.)	Certifikat nr.
	30/1 98.	TL	0336-N2-M-	PwH	~~N M~~

Rekvirent (Entreprenør): Odense Staalshibsværft. Rekv.

Bygherre (Kontrolsted): Lindø.

Emne (Mrk., dim., antal): 1 stk. Svejseprøve nr: 9730.

Tegning nr./Revision:

Materialebetegnelse:

Supplerende oplysninger (uden ansvar, se bagsiden)

Fugeform: [X] I [ ] Y [ ] U [ ] V [X] K [ ] I [ ] Kantsøm [ ] Andet — Overvulst + gennemlæb: [ ] Flad [ ] Alm. [ ] Høj ____ mm — Sidekant/-dybde

Svejsemetode: [ ] 111 Ma.lysbue [ ] 121 Pulversv. [ ] 131 MIG [ ] 135 MAG [ ] 136 Fluxfyldt tråd [ ] 141 TIG — Andre processer: [X] Laser.

Varmebehandlet: [ ] Ja [X] Nej — Overfladens beskaffenhed: [ ] Sleben [X] Jævn [ ] Grov — Tidspunkt for undersøgelse: Timer efter svejsning ____

Undersøgelsesprocedure: [ ] ASME V [ ] ADM-HP 5/3 [ ] BS 6072 [X] OS-QH 414 E.

[ ] Strømmagnetisering — Smax.= ____ mm, Ieff. = ____ Amp. — Apparat nr. ____

[X] Elektromagnet — Type Tiede, Smax.= 100. mm — Apparat nr. EMC 25.

Spolemagnetisering — K (vindinger) = ____, I= ____ Amp. — Strøm: [ ] AC [ ] DC — Apparat nr. ____

[ ] Permanent magnet — S = ____ mm — Magnet nr. ____ — Løftekraft ____

[ ] Prøvebænk — Langsg. ____ Amp — Rundg. ____ Amp. vinc

Elektrodetype: [ ] Stål [ ] Kobber [ ] Andet

Brændpletter slebet: [ ] Ja [ ] Nej

Afmagnetisering: [ ] Ja [X] Nej

Magnetpulver: [X] Vand/sort [ ] opløsning [ ] Vand/fluor. — [ ] Petroleum/sort [ ] Petroleum/fluoresc. [ ] Tørpulver

Baggrund: [X] Hvid [ ] Blankslebet [ ] Ubehandle

Feltstyrke: ____ kA/m, Ørstedmeterværdi ____ Oe — [ ] Andet

Belysning: [X] Hvid [ ] UV-lampe nr. ____

Magnetiseret for: [ ] Langsgående fejl [ ] Tværgående fejl [X] Alle retninger

Kvalitetskrav (Standard): [ ] ASME VIII.1 [ ] ADM-HP 5/3 [ ] DS 412, sømklasse ____ inkl. visuel kontrol [X] Andet: OS-QH 414 E.

Undersøgelsens omfang:

100 % MT/Lt på svejseprøve nr: 9730. begge sider.

Undersøgelsens resultat:

Der er fundet indikationer som angivet på vedlagte bilag.

SV. 2 godkendt i.h.t. kvalitetskrav.

Reparationsopmærkning: [X] På emne [X] På skitse — [ ] Kontrol af opslibning [ ] Kontrol af reparation [ ] Stemplet mrk.: ____

Bilag vedlagt: [X] 1 Stk.

Tekniker (Dato/Underskrift): 30/1 98 FORCE Institutttet F136 Thomas Lund

Att. af rekvirent	Kopi afleveret til	Att. af bygherre/Tilsyn	Kopi afleveret til	Att. af myndighed	Kopi afleveret
Dato/Underskrift		Dato/Underskrift		Dato/Underskrift	

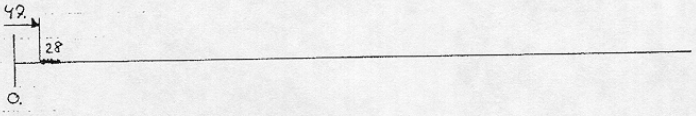
66.D. 04.96

26/01 01 13:29 FAX 45639/2318 OSS PIA 7

FORCE INSTITUTTET

Sag nr./File No.	Udarb. af/Drafted by	Dato/Date	Side/Page
S 722591.	PwH, TLL.	30/1 98.	1/1.

Emne/Object
MT af svejseprøve.



sv. 1.
Længde : 1340 mm.

sv. 2.
Længde : 1287 mm.

d. 30/1 98
FORCE Institutttet
F-1361 Thomas Lund

26/01 01 13:29 FAX 4563972318 OSS PIA 003

Markrapport/Ultralyd svejsesømme

FORCE INSTITUTTET

Division for Inspektion

DANAK Reg. nr.	Sag nr.	Bilag nr.	Rapport nr.	Side af
	S 722597.		U 11.	1/1.
Rekvisitions nr.	**Kontrolperiode/Dato**	**Tekniker (Init.)**	**Certifikat nr.**	**Assistent (Init.)** / **Certifikat nr.**
	1/4 98.	TU	0326N2-1-U	— / -N -1-U

Rekvirent: Odense Stålskibsværft — Entreprenør: Rekv.

Bygherre — Kontrolsted: Lindø.

Emne (Mrk., dim., antal): 1 stk. Svejseprøve nr: 9230. — Tegning nr./Revision — Materialebetegnelse

Fugeform: [X] X [ ] Y [ ] U [ ] V [X] K [ ] I [ ] Andet — Overvulst: [ ] Flad [ ] Alm. [ ] Høj — Sidekærv: ____ mm

Svejsemetode: [ ] Ma, lysbue 111 [ ] Pulversv 121 [ ] MIG 131 [ ] MAG 135 [ ] Fluxtrådet 136 [ ] TIG 141 — Andet: [X] Laser

Varmebeh.: [ ] Ja [X] Nej — Overflade af svejsn.: [ ] Slebet [ ] Jævn [ ] Grov — Timer efter svejsning

Unders. procedure/klasse: [ ] ASME V [ ] IIW 527-76 [ ] ADM-HP 5/3, Klasse ____ — Andet: [X] OS-QH 473.

Apparattype: [ ] USL 31/32 [X] USK 7 D [ ] USK 7 — Andet — Apparat Reg. nr.: ULR 13.

		A:	B:		C:	D:	E:
Normal hoved	Type	MSEB 45		Vinkelhoved Type	MWB 70		
	Reg. nr./Frekv.	44. / 4. MHz	/ MHz	Reg. nr. / Sandvinkel	131. / 70. °	/ °	/ °
Fejlsomhedsindstilling	Grundindstilling ~ ref. kurve	1 ekko dB	dB	Grundindstilling ~ ref. kurve	42 dB	dB	dB
	Overføringskorrektion	1 dB	dB	Overføringskorrektion	4 dB	dB	dB
	Afsøgningstillæg	F.S.H. dB	dB	Afsøgningstillæg	6 dB	dB	dB
	Sum	+6. dB	dB	Sum	52. dB	dB	dB

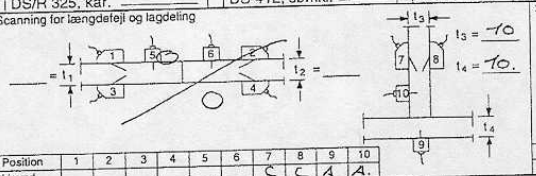
DAC: Hul ø = 1,2 mm, Skalalængde = 0-100 mm AVG: Skive ø = ____ mm, Skalalængde = ____ mm

[X] V1-Reg. nr. 24 [X] V2-Reg. nr. 49 [ ] TIF-blok Reg. nr. ____ — Andet: [X] ABS Blok. R4

[X] Operatørkontrol af lydhoveder udført

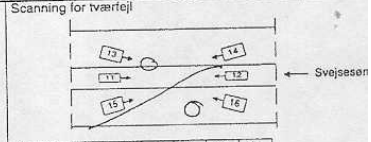
Kvalitetskrav: [ ] DS/R 325, kar. ____ [ ] DS 412, sømkl. ____ [ ] ASME VIII.1 [ ] ASME B 31.3 [ ] ADM-HP 5/3 — Andet: [X] OS-QH 4

Scanning for længdefejl og lagdeling



Position	1	2	3	4	5	6	7	8	9	10
Hoved							C	C	A	A

Scanning for tværfejl



Position	11	12	13	14	15	16
Hoved						

Undersøgelsens omfang

100% UT/L på 1 stk Svejseprøve (Længde 1265 mm)

Undersøgelsens resultat

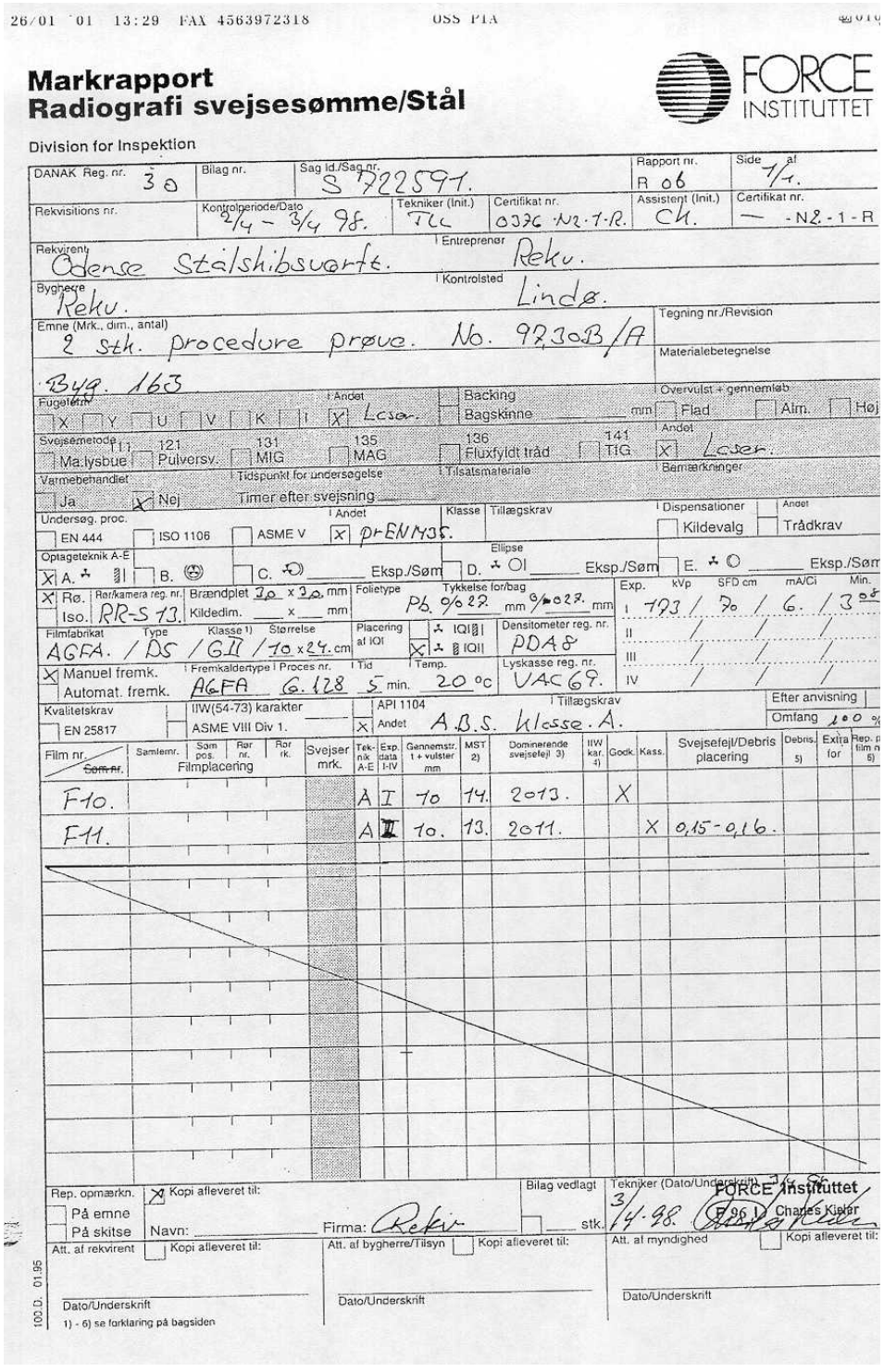
Godkendt i.h.t. kvalitetskrav.

Rep. opmærkning: [ ] På emne [ ] På skitse — [ ] Kontrol af opslibning [ ] Kontrol af reparation [ ] Stemplet ____ — Bilag vedlagt: [ ] 1 stk.

Tekniker (Dato/Underskrift): 1/4 98 FORCE Institutttet (F1361) Thomas Lund

Att. af rekvirent	Kopi afleveret til:	Att. af bygherre/Tilsyn	Kopi afleveret til:	Att. af myndighed	Kopi afleveret til
Dato/Underskrift		Dato/Underskrift		Dato/Underskrift	

151.D. 02.94



26/01 '01 13:29 FAX 4563972318 OSS PIA 010

Markrapport
Radiografi svejsesømme/Stål

FORCE INSTITUTTET

Division for Inspektion

DANAK Reg. nr. 30 | Bilag nr. | Sag id./Sag nr. S 722591. | Rapport nr. R 06 | Side 1/1. af

Rekvisitions nr. | Kontrolperiode/Dato 2/4 - 3/4 98. | Tekniker (Init.) TLL | Certifikat nr. 0336-N2-1-R. | Assistent (Init.) CH. | Certifikat nr. — -N2-1-R

Rekvirent Odense Stålskibsværft. | Entreprenør Reku.

Bygherre Reku. | Kontrolsted Lindø.

Emne (Mrk., dim., antal) 2 stk. procedure prøve. No. 9230B/A | Tegning nr./Revision

Materialebetegnelse

Byg. 163

Fugeform: [X] X [ ] Y [ ] U [ ] V [ ] K [ ] I [X] Andet Laser. | Backing: [ ] Bagskinne ___ mm | Overvulst + gennemlab: [ ] Flad [ ] Alm. [ ] Høj

Svejsemetode: [ ] Ma.lysbue [ ] 121 Pulversv. [ ] 131 MIG [ ] 135 MAG [ ] 136 Fluxfyldt tråd [ ] 141 TIG [X] Andet Laser.

Varmebehandlet: [ ] Ja [X] Nej | Tidspunkt for undersøgelse: Timer efter svejsning ___ | Tilsatsmateriale | Bemærkninger

Undersøg. proc.: [ ] EN 444 [ ] ISO 1106 [ ] ASME V [X] Andet prEN1435. | Klasse | Tillægskrav | Dispensationer: [ ] Kildevalg | Andet: [ ] Trådkrav

Optageteknik A-E: [X] A. [ ] B. [ ] C. ___ Eksp./Sem [ ] D. Ellipse ___ Eksp./Sem [ ] E. ___ Eksp./Sem

[X] Rø. [ ] Iso. | Rør/kamera reg. nr. RR-S 13. | Brændplet 10 x 10 mm | Kildedim. ___ x ___ mm | Folietype Pb. 0,02 mm 0,02 mm | Tykkelse for/bag

Exp.	kVp	SFD cm	mA/Ci	Min.
I 193	70	6.	3	5
II				
III				
IV				

Filmfabrikat AGFA. | Type D5 | Klasse 1) G II | Størrelse 10 x 24. cm | Placering af IQI [X] IQI-B [ ] IQI-W | Densitometer reg. nr. PDA 8

[X] Manuel fremk. [ ] Automat. fremk. | Fremkaldertype | Proces nr. AGFA G.128 | Tid 5 min. | Temp. 20 °C | Lyskasse reg. nr. VAC 69.

Kvalitetskrav: [ ] EN 25817 | IIW(54-73) karakter ___ | [ ] ASME VIII Div 1. | [X] API 1104 | Tillægskrav: [X] Andet A.B.S. klasse A. | Efter anvisning | Omfang 100 %

Film nr. / ~~Som nr.~~	Samlenr.	Filmplacering: Søm pos.	Rør nr.	Rør rk.	Svejser mrk.	Tek-nik A-E	Exp. data I-IV	Gennemstr. t + vulster mm	MST 2)	Dominerende svejsefejl 3)	IIW kar. 4)	Godk.	Kass.	Svejsefejl/Debris placering	Debris. 5)	Extra for	Rep. på film nr. 6)
F-10.						A	I	10	14.	2013.			X				
F-11.						A	II	10.	13.	2011.			X	0,15-0,16.			

Rep. opmærkn. [ ] På emne [ ] På skitse | [X] Kopi afleveret til: Navn: ___ Firma: Reku | Bilag vedlagt ___ stk. | Tekniker (Dato/Underskrift) 3/4-98. FORCE Institutttet (F96) Charles Kiær

Att. af rekvirent [ ] Kopi afleveret til: Dato/Underskrift

Att. af bygherre/Tilsyn [ ] Kopi afleveret til: Dato/Underskrift

Att. af myndighed [ ] Kopi afleveret til: Dato/Underskrift

100.D. 01.95

1) - 6) se forklaring på bagsiden

26/01 '01 13:29 FAX 4563972318 OSS PIA

DATASHEET WELDING LABORATORY

REPORT NO.:	PAGE:
97.3o	1o.

TESTPIECES:

MARKING	DIMENSIONS
WELDING LAB. NO. 97.3o ~ WPS 751.oo2	T- 15o·1o·9oo+ 15o·1o·1285 mm

BREAKTESTING:

	DIMENSIONS	FIRST SIDE	SEC. SIDE	REMARKS
1	T- 15o·1o·5o+ 15o·1o·5o	GRINDED	BENDED	OK
2	T- 15o·1o·5o+ 15o·1o·5o	BENDED	GRINDED	OK
3	T- 15o·1o·5o+ 15o·1o·5o	GRINDED	BENDED	OK
4	T- 15o·1o·5o+ 15o·1o·5o	BENDED	GRINDED	OK

IMPACT TESTING (CHARPY V)

SIZE OF TESTPIECES 7,5·1o·55 mm. C_V= 5/6·47 J. = 39 J. BY +2o

TESTING TEMP. +2o°C		TESTING TEMP. + 10°c		TESTING TEMP. + 10°c		TESTING TEMP. + 10°c	
NO.	IMPACT STRENGTH	NO.	IMPACT STRENGTH	NO.	IMPACT STRENGTH	NO.	IMPACT STRENGTH
1	126						
2	91						
3	95						
MEAN	1o4						
CENTER OF WELD		FUSION LINE		FL + 2		FL + 5	

BEND TESTING

NO.	DIMENSIONS	FORMER/BEND ANGEL	REMARKS
4	1o·2o· mm	35 mmØ 17o °	NO VISIBLE CRACKS.
			ELONGATION ~ 22 %

MACRO ETCHING

CROSS SECTION 1	CROSS SECTION 2	CROSS SECTION 3
OK.	OK.	OK.

LINDØ D.: 98o4o3. SIGN.: [signature]

ABS ODENSE · Lloyd's Register of Shipping ODENSE 15/4 98

DATASHEET WELDING LABORATORY	97.3o	11.

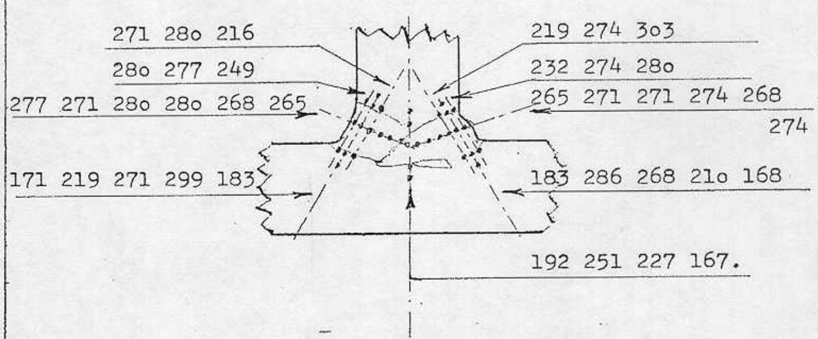
TESTPIECES:

MARKING	DIMENSIONS
WELDING LAB. NO. 97.3o WPS 751.oo2	T-15o·1o·1285 + 15o·1o·1285

BREAKING-TEST OF FILLET WELDING.

DIMENSIONS	First side	Sec. side	Results.
1.	Machined	Fractured	
2.	Fractured	Machined	
3.	Machined	Fractured	
4.	Fractured	Machined	

HARDNESS TEST. VICKERS H_V 5.



MACRO ETCHING

CROSS SECTION 1+2+3	
Ok. Ok. Ok.	See photo, page: 13.

LINDØ D: o4o698. Sign.: [signature]

980303 / MH.

26/01 '01 13:29 FAX 4563972318 OSS PTA 013

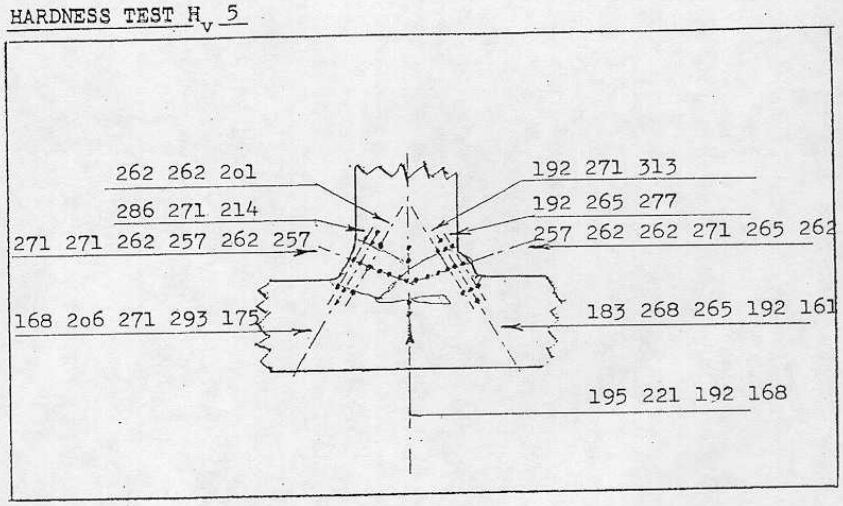
DATASHEET WELDING LABORATORY

REPORT NO.	PAGE
97.3o	12.

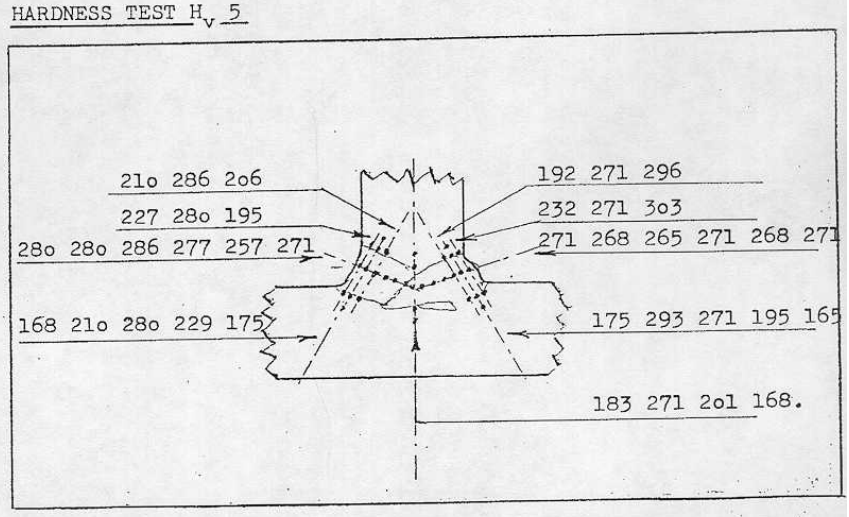
TESTPIECES:

MARKING	DIMENSIONS
WELDING LAB. NO. 97.3o WPS 751.oo2	T-15o·1o·1285 + 15o·1o·1285mm

HARDNESS TEST H_V 5



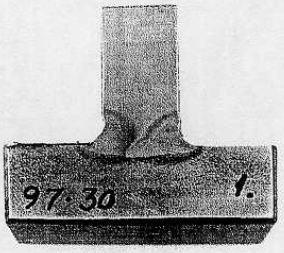
HARDNESS TEST H_V 5

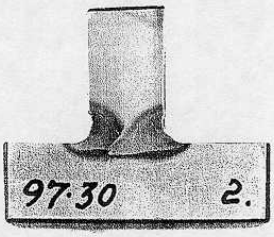


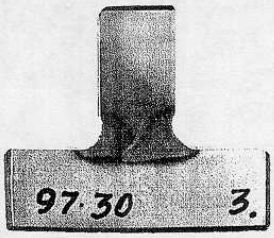
980303 / MH.

Lindø D.: o6o498 Sign.:

DATASHEET WELDING LABORATORY	REPORT NO.:	PAGE:
	97.30	13







CUTTED AND ETCHED SECTIONS.

Annex D

Results of Magnetic Powder Inspection

This Annex comprises the test report from the Force Institutes regarding magnetic powder inspection of the test pieces, which were welded for the experimental part of the present work.

Markrapport
Magnetpulverprøvning

FORCE INSTITUTTET

Division for Inspektion

DANAK Reg. nr.	Bilag nr.	Sag id./Sag nr.	Rapport nr.	Side af
30		S722629	M-10	1 / 1

Rekvisitions nr.	Kontrolperiode/Dato	Tekniker (Init.)	Certifikat nr.	Assistent (Init.)	Certifikat nr.
	1/4-98 - 2/4-98	CK	0038-N2-M-	PWH	-N -M-

Rekvirent: ODENSE STAALSKIBSVÆRFT A/S — Entreprenør: REKVIRENT

Bygherre: REKVIRENT — Kontrolsted: LINDØ

Emne (Mrk., dim., antal): MPI AF 46 STK. LASERSVEJSTE SVEJSEPRØVER — Tegning nr./Revision: Skitser.

B/G. 164. — Materialebetegnelse: Fe

Supplerende oplysninger (uden ansvar, se bagsiden)

Fugeform: [ ] X [ ] Y [ ] U [ ] V [ ] K [ ] I [X] Kantsøm — Andet: — Overvulst < gennemlob: Flad [X] Alm. [ ] Høj — Sidekærv dybde: ___ mm

Svejsemetode: [ ] 111 Må lysbue [ ] 121 Pulversv. [ ] 131 MIG [ ] 135 MAG [ ] 136 Fluxfyldt tråd [ ] 141 TIG — Andre processer: [X] Laser

Varmebehandlet: [ ] Ja [X] Nej — Overfladens beskaffenhed: [ ] Sleben [X] Jævn [ ] Grov — Tidspunkt for undersøgelse: Timer efter svejsning ___

Undersøgelsesprocedure: [ ] ASME V [ ] ADM-HP 5/3 [ ] BS 6072 — Andet: [X] OS/QH - 414E

Strømmagnetisering: [ ] — Smax.= ___ mm, Iakt. = ___ Amp. — Apparat nr. ___

Elektromagnet: [X] — Type TIEDE, Smax.= 100 mm, Apparat nr. EMC-25

Spolemagnetisering: [ ] — K (vindinger) = ___, I= ___ Amp., Strøm: [ ] AC [ ] DC, Apparat nr. ___

Permanent magnet: [ ] — S = ___ mm, Magnet nr. ___, Løftekraft ___

Prøvebænk: [ ] — Langsg. ___ Amp., Rundg. ___ Amp. vind.

Elektrodetype: [ ] Stål [ ] Kobber — Andet: [ ] — Brændpletter slebet: [ ] Ja [ ] Nej — Almagnetisering: [ ] Ja [X] Nej

Magnetpulver: [X] opløsning — Vand/sort [X] Vand/fluor. — [ ] Petroleum/sort [ ] Petroleum/fluoresc. — [ ] Tørpulver

Baggrund: [ ] Hvid [ ] Blankslebet [X] Ubehandlet

Feltstyrke: ___ kA/m, Ørstedmeterværdi ___ Oe — Andet: [X] Castrol Silver — Belysning: [ ] Hvid [X] UV-lampe nr. UVG 5

Magnetiseret for: [ ] Langsgående fejl [ ] Tværgående fejl [X] Alle retninger

Kvalitetskrav (Standard): [ ] ASME VIII.1 [ ] ADM-HP 5/3 — DS 412, sømklasse ___ [ ] inkl. visuel kontrol — Andet: [X] OS/QH 414E

Undersøgelsens omfang: 100% MPI AF 46 STK SVEJSEPRØVER. SE VEDLAGTE BILAG FOR PRØVE-NR OG SVEJSE LÆNGDER.

Undersøgelsens resultat: SE VEDLAGTE BILAG FOR RESULTAT.

Reparationsopmærkning: [X] På emne [X] På skitse — Kontrol af opslibning [ ] — Kontrol af reparation [ ] — Stemplet mrk.: ___

Bilag vedlagt: [X] 10 Stk.

Tekniker (Dato/Underskrift): 3/4-98 Charles Kieler — E Instituttet Charles Kieler

Att. af rekvirent	Kopi afleveret til	Att. af bygherre/Tilsyn	Kopi afleveret til	Att. af myndighed	Kopi afleveret til
Dato/Underskrift		Dato/Underskrift		Dato/Underskrift	

95 D 04 98

FORCE INSTITUTTET

Rapport. M. 10

Sag nr./File No.	Udarb. af/Drafted by	Dato/Date	Side/Page
S722629	PWH	02.04.1998	BILAG 1

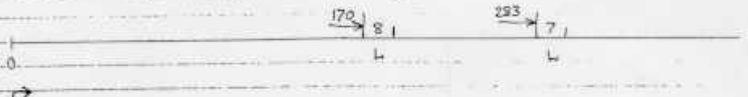
Emne/Object

M.P.I AF LASERSVEJSTE SVEJSEPRØVER

PRØVE NR.: 270398-6B1A SVEJSNINGENS LÆNGDE: 522 m/m.

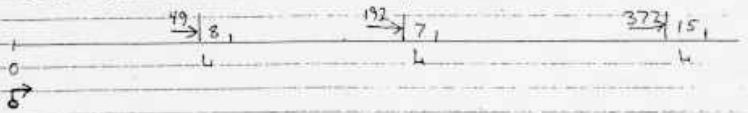
PRØVE NR.: 270398-6B2 SVEJSNINGENS LÆNGDE: 522 m/m.

PRØVE NR.: 270398-6B3 SVEJSNINGENS LÆNGDE: 523 m/m.



PRØVE NR.: 280398-6B4 SVEJSNINGENS LÆNGDE: 564 m/m.

PRØVE NR 280398-6B5 SVEJSNINGENS LÆNGDE: 406 m/m.



M.O. = MANGLENDE OPFYLDNING

R/S = REVNE I F.M START

P = PORER

L = LINEER INDIKATION

OBS: ALLE MÅL ER m/m.

12.93

FORCE INSTITUTTET

Rapport M 10

Sag nr./File No.	Udarb./af/Drafted by	Dato/Date	Side/Page
S722629	PWH	02.04.1998	BILAG 2

Emne/Object

M.P.I AF LASERSVEJSTE SVEJSEPRØVER

PRØVE NR: 280398-6B6 SVEJSNINGENS LÆNGDE: 519 m/m.

0 MPI/OK

PRØVE NR: 280398-6B7 SVEJSNINGENS LÆNGDE: 519 m/m.

0 5→6 R/S; 201→6 L

PRØVE NR: 280398-6B8 SVEJSNINGENS LÆNGDE: 520 m/m.

0 7→6 R/S; 253→10 L

PRØVE NR: 280398-72C5 SVEJSNINGENS LÆNGDE: 521 m/m

0 4→12 R/S; 22→48 m.o.; 295→8 L.

PRØVE NR 280398-73E1 SVEJSNINGENS LÆNGDE: 524 m/m.

0 5→5 R/S; 185→13 m.o.

M.O. = MANGLENDE OPFYLDNING.

R/S = REVNE I F.M START OBS: ALLE MÅL ER m/m.

L = LINEER INDIKATION

58. 12.93

FORCE INSTITUTTET

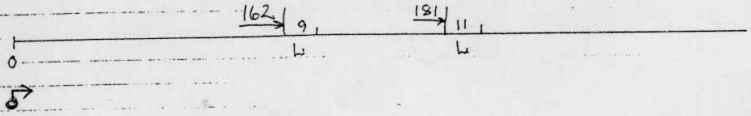
Rapport. M10

Sag nr./File No.	Udarb. af/Drafted by	Dato/Date	~~Side/Page~~
S722629	PWH	02.04.1998	BILAG 3

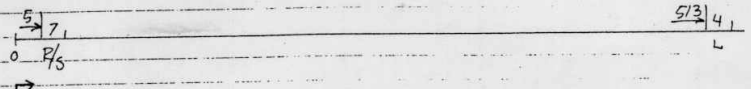
Emne/Object

M.P.I AF LASERSVEJSTE SVEJSEPRØVER

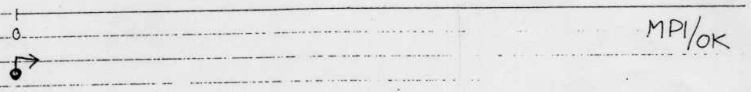
PRØVE NR: 280398 - 73E3 SVEJSNINGENS LÆNGDE: 535 m/m.



PRØVE NR: 290398 - 71D1 SVEJSNINGENS LÆNGDE: 520 m/m.

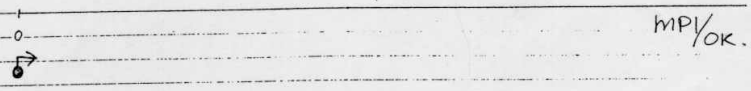


PRØVE NR: 290398 - 71D2 SVEJSNINGENS LÆNGDE: 522 m/m.



MPI/OK

PRØVE NR: 290398 - 71D3 SVEJSNINGENS LÆNGDE: 521 m/m.



MPI/OK.

PRØVE NR. 290398 - 72C1 SVEJSNINGENS LÆNGDE: 478 m/m.

MPI/OK

M.O. = MANGELENDE OPFYLDNING.

R/S = REVNE I F.M START OBS: ALLE MÅL ER m/m

L = LINEER INDIKATION

12.93
98.

FORCE INSTITUTTET

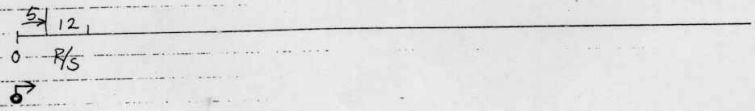
Rapport. M 10

Sag nr./File No.	Udarb. af/Drafted by	Dato/Date	Side/Page
5722629	PWH	02.04.1998	BILAG 4

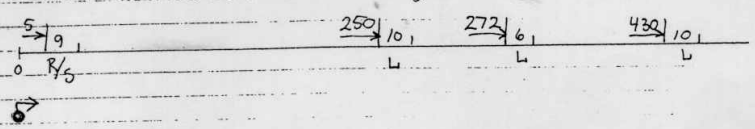
Emne/Object

M.P.I AF LASERSVEJSTE SVEJSEPRØVER

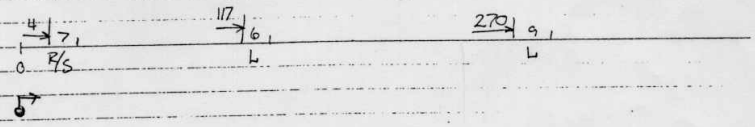
PRØVE NR.: 290398-72C2 SVEJSNINGENS LÆNGDE: 521 m/m.



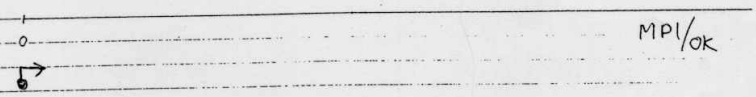
PRØVE NR.: 290398-72C3 SVEJSNINGENS LÆNGDE: 520 m/m.



PRØVE NR.: 290398-72C4 SVEJSNINGENS LÆNGDE: 521 m/m.



PRØVE NR.: 290398-73C1 SVEJSNINGENS LÆNGDE: 522 m/m.



MPI/OK

PRØVE NR 290398-73C2 SVEJSNINGENS LÆNGDE: 521 m/m.

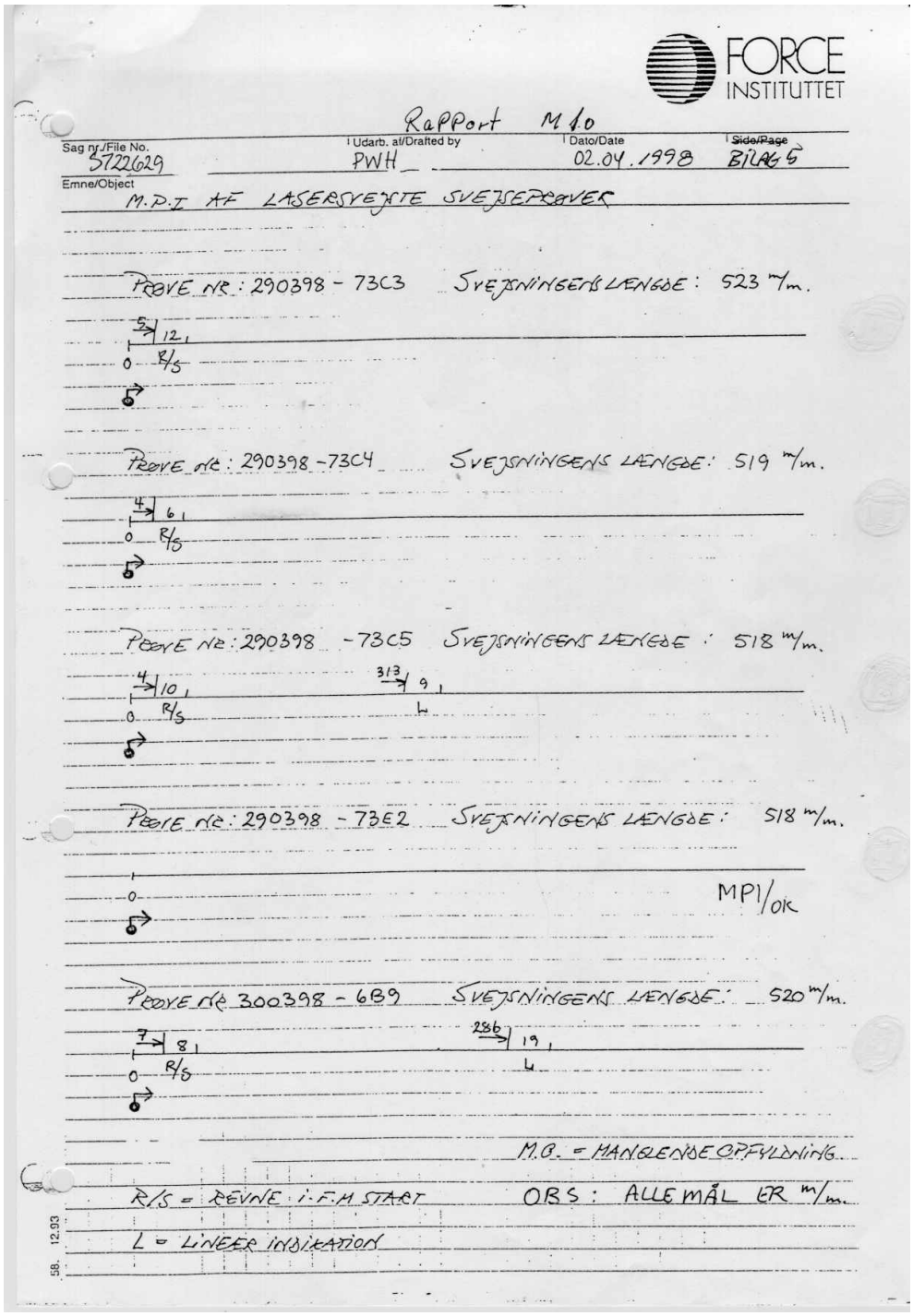
M.O. = MANGELENDE OPFYLDNING

R/S = REVNE I F.M START

OBS: ALLE MÅL ER m/m

L = LINEER INDIKATION

58. 12.93



FORCE INSTITUTTET

RAPPORT M.10

Sag nr./File No. 5722629 | Udarb. af/Drafted by PWH | Dato/Date 02.04.1998 | ~~Side/Page~~ BILAG 5

Emne/Object

M.P.I AF LASERSVEJSTE SVEJSEPRØVER

PRØVE NR.: 290398 - 73C3 SVEJSNINGENS LÆNGDE: 523 m/m.

5 → 12
0 R/S

PRØVE NR.: 290398 -73C4 SVEJSNINGENS LÆNGDE: 519 m/m.

4 → 6
0 R/S

PRØVE NR: 290398 -73C5 SVEJSNINGENS LÆNGDE: 518 m/m.

4 → 10
0 R/S
313 → 9
L

PRØVE NR: 290398 - 73E2 SVEJSNINGENS LÆNGDE: 518 m/m.

0

MPI/OK

PRØVE NR 300398 - 6B9 SVEJSNINGENS LÆNGDE: 520 m/m.

7 → 8
0 R/S
286 → 19
L

M.O. = MANGLENDE OPFYLDNING.

R/S = REVNE I F.M START OBS: ALLE MÅL ER m/m.

L = LINEER INDIKATION

58. 12.93

FORCE INSTITUTTET

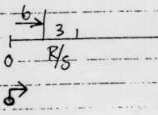
Rapport M.10

Sag nr./File No.	Udarb. af/Drafted by	Dato/Date	~~Side/Page~~
5722629	PWH	02.04.1998	BILAG 6

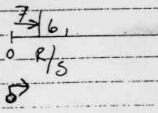
Emne/Object

M.P.I AF LASERSVEJSTE SVEJSEPRØVER

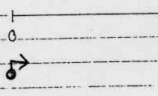
PRØVE NR.: 300398 - 6B10 SVEJSNINGENS LÆNGDE: 475 m/m.



PRØVE NR.: 300398 - 6B11 SVEJSNINGENS LÆNGDE: 473 m/m.

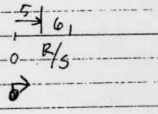


PRØVE NR.: 300398 - 7ID1 SVEJSNINGENS LÆNGDE: 522 m/m.

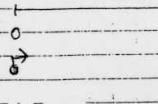


MPI/OK

PRØVE NR.: 300398 - 7ID2 SVEJSNINGENS LÆNGDE: 524 m/m.



PRØVE NR. 300398 - 73F1 SVEJSNINGENS LÆNGDE: 518 m/m.



MPI/OK

M.O. = MANGLENDE OPFYLDNING

R/S = REVNE I F.M START

L = LINEER INDIKATION

OBS: ALLE MÅL ER m/m.

58. 12.93

FORCE INSTITUTTET

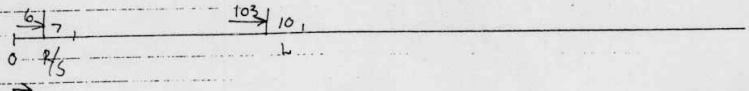
Rapport. M. 10

Sag nr./File No.	Udarb. af/Drafted by	Dato/Date	~~Sider/Page~~
5722629	PWH	02.04.1998	BILAG 7

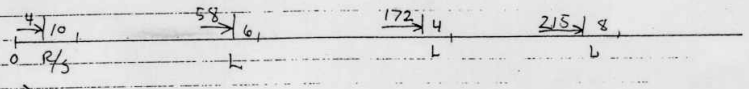
Emne/Object

M.P.I AF LASERSVEJSTE SVEJSEPRØVER

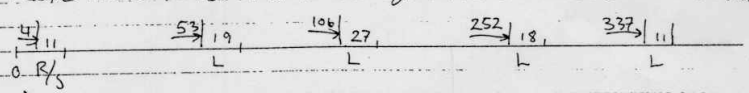
PRØVE NR: 300398 - 73F2 SVEJSNINGENS LÆNGDE: 516 m/m.



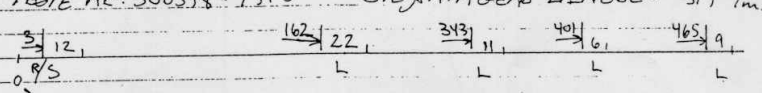
PRØVE NR: 300398 - 73F3 SVEJSNINGENS LÆNGDE: 519 m/m.



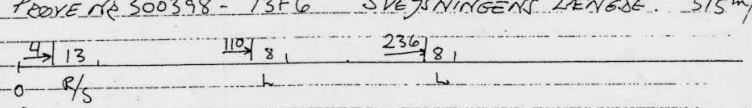
PRØVE NR: 300398 73F4 SVEJSNINGENS LÆNGDE: 517 m/m.



PRØVE NR: 300398 - 73F5 SVEJSNINGENS LÆNGDE: 519 m/m.



PRØVE NR 300398 - 73F6 SVEJSNINGENS LÆNGDE: 515 m/m.



M.O. = MANGLENDE OPFYLDNING.

R/S = REVNE I F.M START

OBS: ALLE MÅL ER m/m.

L = LINEER INDIKATION

12.03
58

FORCE INSTITUTTET

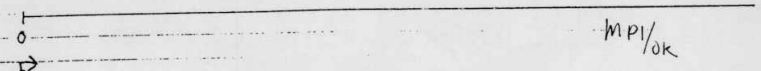
Rapport M. 10

Sag nr./File No.	Udarb. af/Drafted by	Dato/Date	Side/Page
5722629	PWH	02.04.1998	BILAG 8

Emne/Object

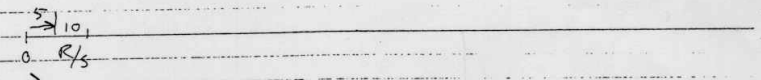
M.P.I AF LASERSVEJSTE SVEJSEPRØVER

PRØVE NR: 300398-73F7 SVEJSNINGENS LÆNGDE: 524 m/m.

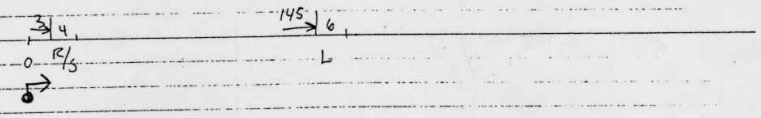


PRØVE NR: 300398-73G1 SVEJSNINGENS LÆNGDE: 528

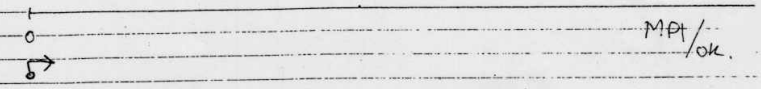
PRØVE NR: 300398-73G2 SVEJSNINGENS LÆNGDE: 525 m/m.



PRØVE NR: 310398-72B1 SVEJSNINGENS LÆNGDE: 523 m/m.



PRØVE NR 310398-72B2 SVEJSNINGENS LÆNGDE: 523 m/m.



M.O. = MANGLENDE OPFYLDNING.

R/S = REVNE I F.M START

OBS: ALLE MÅL ER m/m.

L = LINEER INDIKATION

58. 12.93

FORCE INSTITUTTET

Rapport. M. 10

Sag nr./File No.	Udarb. af/Drafted by	Dato/Date	Side/Page
S722629	PWH	02.04.1998	BILAG 9

Emne/Object

M.P.I AF LASERSVEJSTE SVEJSEPRØVER

PRØVE NR.: 310398-73B1 SVEJSNINGENS LÆNGDE: 525 m/m.

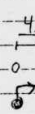
MPI/OK.

PRØVE NR.: 310398-73B2 SVEJSNINGENS LÆNGDE: 522 m/m.

MPI/OK.

PRØVE NR.: 310398-73B3 SVEJSNINGENS LÆNGDE: 526

PRØVE NR.: 310398-73B4 SVEJSNINGENS LÆNGDE: 527 m/m.



PRØVE NR 310398-73B4 SVEJSNINGENS LÆNGDE: 523 m/m

MPI/OK.

M.O. = MANGLENDE OPFYLDNING.

R/S = REVNE I F.M START

O.B.S. ALLE MÅL ER m/m

L = LINEER INDIKATION

12.93

58.

FORCE INSTITUTTET

Rapport. M. 10

Sag nr./File No.	Udarb. af/Drafted by	Dato/Date	Side/Page
S4722629	PWH	02.04.1998	BILAG 10.

Emne/Object

M.P.I AF LASERSVEJSTE SVEJSEPRØVER

PRØVE NR: ~~R-1-SA~~ Z203 15 SVEJSNINGENS LÆNGDE: 522 m/m.

MPI/OK.

~~PRØVE NR:~~ ~~SVEJSNINGENS LÆNGDE:~~

~~PRØVE NR:~~ ~~SVEJSNINGENS LÆNGDE:~~

~~PRØVE NR:~~ ~~SVEJSNINGENS LÆNGDE:~~

~~PRØVE NR~~ ~~SVEJSNINGENS LÆNGDE:~~

M.O. = MANGLENDE OPFYLDNING.

R/S = REVNE I F.M START

L = LINEER INDIKATION O.B.S.: ALLE MÅL ER m/m.

12.93