Olivier Marchal – Nicolas Ponsart

March – June 2001

Real-Time Visual Servoing Appendices

Supervised by:
Dr. Harvey Lipkin, Associate Professor at Georgia Institute of Technology
Thierry Nowak, Electrical Engineering Professor at Ecole Nationale d'Ingenieurs de Metz

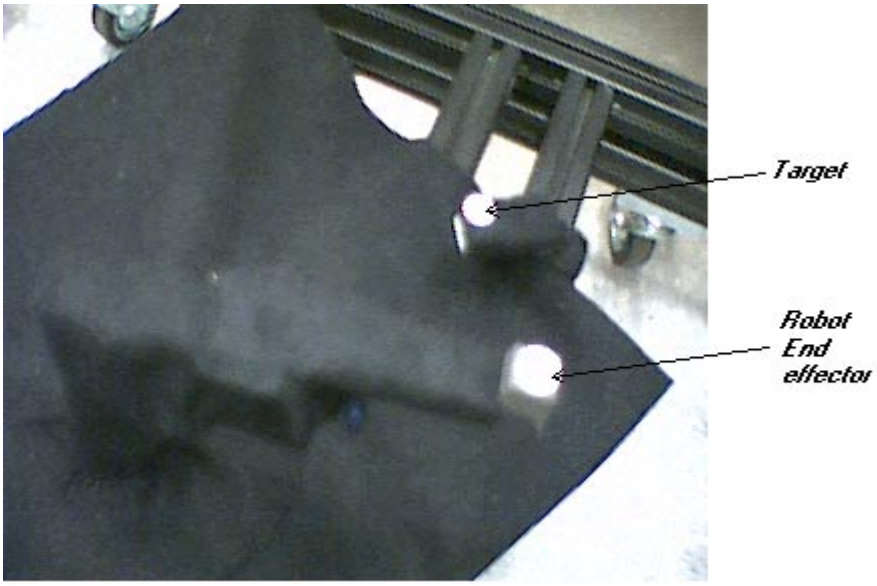
Final-Year Project 2001

Appendices Contents

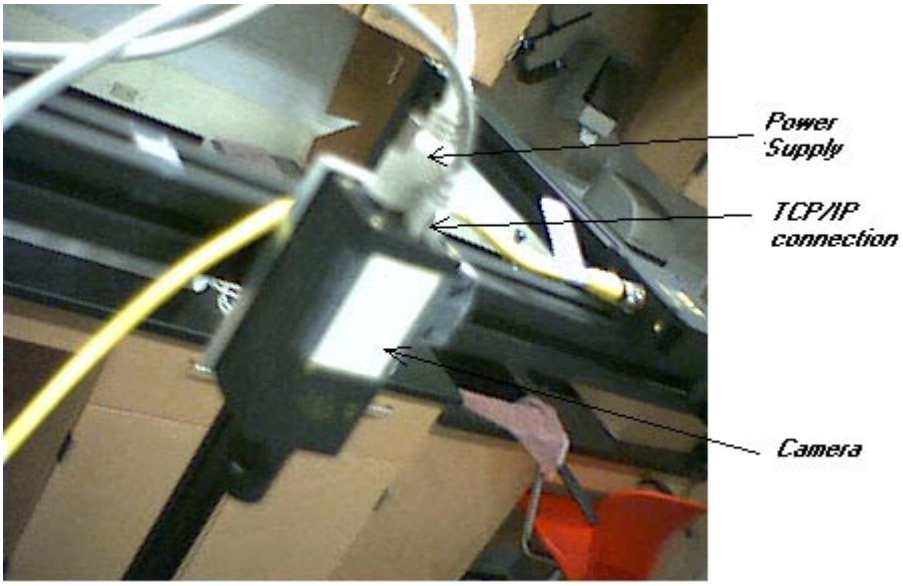
Appendix 1

Hardware configuration of our system

Here is the structure with our target and our robot



Here is our camera, which is mounted on the same structure



The distance between the robot and the camera is of 1200 mm.

Appendix 2
Visual Basic Program

Here is our Visual Basic program. More interesting parts are subroutines «sckDVT_DataArrival », « Start » and « Timer ».

```
'********************************************************************************
'Force explicit declaration of all variables
'********************************************************************************

Option Explicit

Dim strMoveJ1 As String          'New joint angle increment retrieved from Matlab
Dim strMoveJ2 As String


Dim FirstMod As Long             'initial switch modification count (mod)
Dim CurrMod As Long              'current switch modification count (mod)
Dim InitialTime As Double        'system Time when parking begins
Dim totaltime As Long            'total system Time (in seconds) parked
Dim EiOK As Long                 '"Ei Responding" indicator
Dim SwitchValue As Long          'the value of the switch
Dim ErrMsg As String             'temporary message variable
Dim RetryCancel As Byte          'message box result holder

Dim X As Variant                 'For error displaying
Dim Msg As String
Dim DescripSiz As Long
Dim RetSiz As Long
Dim ErrDescrip As String         'Create string to hold error description

Dim RobotName As String          'Create string to hold our robot's name.
Dim LptPort As Long              'LPT port to which a Robix Rascal Ei is connected
Dim Irq As Long                  'IRQ associated with the above LPT port
Dim NameSize As Long             'size (in bytes) of robot name
Dim RetSize As Long              'number of bytes of robot name copied by DLL
Dim ServoCount As Long           'number of servos assigned to robot
Dim AuxOutputCount As Long       'number of aux outputs assigned to robot
Dim AdcValue

Dim Joint1 As String             'set two variables to initialize the test
Dim Joint2 As String
```

```
'tells if Matlab is launched and is ready to calculate new joint angles increments
Dim MatlabReady As Long
'permits a new calculation step for Matlab. New picture
Dim NextCalculation As Integer
'tells the application started. To avoid errors due to timer before clicking on "Start"
Dim ProgramRunning As Integer

'create an object variable that holds a reference to an excel application
Dim XLApp As New Excel.Application

Dim strData As String              'holds data after retrieval from the Winsock object

Dim Script As String               'script to make the robot move according to the desired
joint angles
Dim RetCode As Long                 'scratch variable; records a return value
Dim ConfigPath As String           'path to configuration file, hardware and software settings
Dim EiHandle As Long                'Ei handle
Dim RobotHandle As Long             'Robot handle
```

Private Sub Restart_Click()

```
 'Restart robot. Note return code.
    RetCode = rbxRobotRestart(RobotHandle)

    'Check return code
    If RetCode <> RBX_E_SUCCESS Then
        DisplayRobixError "Error restarting robot.", RetCode
        Exit Sub
    End If
```

End Sub

```
Private Sub sckDVT_DataArrival(ByVal bytesTotal As Long)

'*************************************************************************
' This subroutine executes with every arriving data packet
' at the TCP port specified by the Winsock control
'*************************************************************************
' retrieves the string and stores in strData
sckDVT.GetData strData, vbString


'Send strData to spreadsheet
XLApp.Cells(3, 2) = strData


End Sub


Private Sub Start_Click()

'Tell the timer to start achieving commands
ProgramRunning = 1


'*************************************************************************
' This will allow to make a link with an Excel Spreadsheet, which we need
' to communicate with Matlab
'*************************************************************************

'Set object variable to reference an existing Excel file & open it. This path may change to
fit your settings
XLApp.Workbooks.Open ("C:\My Documents\Nicolas\PFE\Global Visual Servoing
System\Excel\Global_Spreadsheet.xls")

'makes Excel spreadsheet visible if desired. (Not Required)
XLApp.Visible = True

'Initialize NextCalculation to 1 so Matlab does the first calculation
NextCalculation = 1
XLApp.Cells(18, 7) = NextCalculation

'Reset MatlabReady
XLApp.Cells(18, 5) = 0

'Reset the strMoveJ1 and strMoveJ2 to 0 to be ready for the next application launching
XLApp.Cells(20, 2) = 101.27
XLApp.Cells(20, 3) = 0
```

```
'*************************************************************************
' Use the TCP/IP protocol to retrieve data from Framework
'*************************************************************************

' Invoke the connect Method from the Winsock object
' this establishes the connection according to RemoteHost
' and RemotePort properties
sckDVT.Connect

End Sub

Private Sub Stop_Click()

'tells the timer to stop executing commands
ProgramRunning = 0

' Disconnect from the RemoteHost and RemotePort
sckDVT.Close

'Save spreadsheet (Not required)
XLApp.ActiveWorkbook.Save


'close Excel
XLApp.Workbooks.Close

End Sub
```

```
Private Sub Test_Click()

'*************************************************************************
'For test purpose only
'Check if the robot moves correctly with a single test script
'*************************************************************************

' make a single movement. For test purpose
Joint1 = -500
Joint2 = -600

'cancatenation of the script
Script = "Move 1 by " + Joint1 + ", 2 by " + Joint2

'Script = "Move 1 by 600, 2 by -600"
RetCode = rbxScriptExecute(RobotHandle, Script)

'Check return code
If RetCode <> RBX_E_SUCCESS Then
   DisplayRobixError "Error moving robot.", RetCode
   Exit Sub
End If

End Sub



Private Sub Exit_Click()

   'ends the program
   Unload Me

End Sub
```

```
'*************************************************************************
'This subroutine is executed when the main form is loaded. In here we
'specify our program variables.
'*************************************************************************

Private Sub Form_Load()
  'A Robix Rascal Configuration (.rbc) file contains information about Ei's
  'and robots. Configuration files are typically generated using the Robix
  'Rascal Software (RascalGUI.exe).

  'IMPORTANT!
  'Three configuration files are supplied with this program:
  ' VB_Config_L1i7.rbc - Ei connected to LPT1 using IRQ7
  ' VB_Config_L1i5.rbc - Ei connected to LPT1 using IRQ5
  ' VB_Config_L2i5.rbc - Ei connected to LPT2 using IRQ5
  '
  'Please specify the appropriate configuration file below according to
  'your hardware settings. If none of the supplied configuration files
  'match your hardware settings, use the Rascal software to create and save
  'a working configuration file and then specify that configuration
  'file below. See the Rascal help system for further discussion of hardware
  'settings.
  'We are using Lpt1 and IRQ 7. We had to reserve IRQ 7 and reboot the PC
  'because of conflicts with our sound card that was previously using IRQ 7

  'Set program variables
  EiHandle = 0     'Ei handle
  RobotHandle = 0  'robot handle

  'Configuration file path
  'You may need to change this path to adapt this program to your application
  ConfigPath = "C:\My Documents\Nicolas\PFE\Global Visual Servoing System \Visual
Basic\VB_Config_L1i7.rbc"


  End Sub
```

```
'*************************************************************************
'This subroutine is executed when the main form becomes active. In here we
'initialize the Rascal DLL (once only).
'*************************************************************************

Private Sub Form_Activate()

  '*********************************
  '*** Initialize the Rascal DLL ***
  '*********************************

  'We do this here instead of in Form_Load() so that an icon will
  'be present during initialization. This helps prevent users
  'from "losing" any dialog messages that may be generated during
  'DLL initialization.

  'Make sure we only initialize the DLL once (when the form first
  'becomes active.
  Static DoThisOnce As Boolean

  If DoThisOnce = False Then
    'Verify DLL version
    RetCode = rbxDLLVersionVerify()
    If RetCode <> RBX_E_SUCCESS Then
      DisplayRobixError "Error verifying DLL version", RetCode
      End
    End If

    'Check DLL functions
    RetCode = rbxDLLFunctionsCheck()
    If RetCode <> RBX_E_SUCCESS Then
      DisplayError "Error checking DLL functions", RetCode
      End
    End If

    DoThisOnce = True
  End If

End Sub
```

```
'*************************************************************************
'This subroutine is executed when the main form is unloaded. In here we
'clear any configuration that may be loaded into the DLL.
'*************************************************************************
Private Sub Form_Unload(Cancel As Integer)

  'Clear the current configuration. Note return code.
  RetCode = rbxConfigClear()

  'Check return code
  If RetCode <> RBX_E_SUCCESS Then
    DisplayRobixError "Error clearing configuration.", RetCode
    Exit Sub
  End If

End Sub

'*************************************************************************
'This subroutine is executed when the button labeled "Load Configuration"
'is clicked. This subroutine loads an existing Robix Rascal Configuration
'into the DLL, displays some information about that configuration, and
'assigns a script to an existing robot.
'*************************************************************************
Private Sub bnLoadConfiguration_Click()


  '*************************
  '*** Load Configuration ***
  '*************************

  RetCode = rbxConfigFileLoad(ConfigPath)

  'Check return code
  If RetCode <> RBX_E_SUCCESS Then
    DisplayRobixError "Error loading configuration file.", RetCode
    Exit Sub
  End If
```

```
'************************************
'*** Get configuration information ***
'************************************

'Now that the configuration is successfully loaded, we need to acquire
'handles to the existing Ei(s) and robot(s). These handles will be
'needed for various function calls later. In this demo, we assume that
'only one Ei and only one robot exists.

'Get handle of first Ei
RetCode = rbxLptEiHandleGet(1, EiHandle)

'Check return code
If RetCode <> RBX_E_SUCCESS Then
  DisplayRobixError "Error getting Ei handle.", RetCode
  Exit Sub
End If

'Make sure we have a valid handle. Note that handle is zero if Ei does not
'exist. See RascalDLLDoc.txt - rbxEiHandleGet() for further documentation.
If EiHandle = 0 Then
  DisplayError "Ei does not exist."
  Exit Sub
End If

'Get Ei port information
RetCode = rbxLptEiPortInfoGet(EiHandle, LptPort, Irq)

'Check return code
If RetCode <> RBX_E_SUCCESS Then
  DisplayRobixError "Error getting Ei port information.", RetCode
  Exit Sub
End If

'Create a message with the info we've obtained
lbEiInfo.Caption = "Ei enabled. LPT" & LptPort & " - IRQ" & Irq


'Get handle of first robot
RetCode = rbxRobotHandleGet(1, RobotHandle)

'Check return code
If RetCode <> RBX_E_SUCCESS Then
  DisplayRobixError "Error getting robot handle.", RetCode
  Exit Sub
End If
```

```
'Make sure we have a valid handle. Note that handle is zero if robot does not
'exist. See RascalDLLDoc.txt - rbxRobotHandleGet() for further documentation.
If RobotHandle = 0 Then
  DisplayError "Robot does not exist."
  Exit Sub
End If

'Get the size of our robot's name so we know how big our string
'will need to be in order to hold the name. Note return code.
RetCode = rbxRobotNameSizeOf(RobotHandle, NameSize)

'Check return code
If (RetCode <> RBX_E_SUCCESS) Then
  DisplayRobixError "Error getting size of robot name.", RetCode
  Exit Sub
End If

'IMPORTANT:
'Some Rascal DLL functions fill an application-provided string with
'characters, such as rbxRobotNameGet() below. Such functions expect a
'string large enough to hold all of the characters. In order to dynamically
'create strings of appropriate size, VB applications must create a string
'and fill it with the required amount of characters as shown below.
'If your application passes a string of incorrect size, you may receive a
'null-pointer error code, an access violation may occur, or some other
'undesirable activity may occur.


'Make sure RobotName is large enough to hold the robot's name by filling it
'with the required number of characters. (NameSize was acquired by calling
'rbxRobotNameSizeOf() above.) Note that any character can be used
'here; we just use zero for simplicity.
RobotName = String(NameSize, "0")

'Get our robot's name. Note return code.
RetCode = rbxRobotNameGet(RobotHandle, RobotName, NameSize, RetSize)

'Check return code
If RetCode <> RBX_E_SUCCESS Then
  DisplayRobixError "Error getting robot name.", RetCode
  Exit Sub
End If

'Now let's see how many servos and aux outputs are assigned to our robot
```

```
'Get our robot's servo count. Note return code.
RetCode = rbxRobotServoCountGet(RobotHandle, ServoCount)

'Check return code
If RetCode <> RBX_E_SUCCESS Then
  DisplayRobixError "Error getting robot's servo count.", RetCode
  Exit Sub
End If

'Get our robot's aux output count. Note return code.
RetCode = rbxRobotAuxOutputCountGet(RobotHandle, AuxOutputCount)

'Check return code
If RetCode <> RBX_E_SUCCESS Then
  DisplayRobixError "Error getting robot's aux output count.", RetCode
  Exit Sub
End If

'Display robot info in labels on the main form
lbRobotInfo.Caption = "Robot built."
lbRobotName.Caption = "Name: " & RobotName
lbRobotServos.Caption = "Servos: " & ServoCount
lbRobotAuxOuts.Caption = "AuxOutputs: " & AuxOutputCount



'**********************
'*** Restart robot ***
'**********************

'We have now "built" a "logical" robot of some servo and aux outputs
'by loading the config file, but a "logical" robot could also be built
'piece by piece using command such as rbxEiEnable(), rbxRobotCreate(),
'rbxRobotServoAssign(), and rbxRobotAuxOutputAssign(). In any case,
'after a "logical" robot is "built", the physical Ei's that it uses must
'be initialized. Failure to restart the robot before use may cause its
'servos to behave erratically.

'When a robot is restarted, it will quickly come to "attention" by sending
'each servo to its initial position (initpos).

'Restart robot. Note return code.
RetCode = rbxRobotRestart(RobotHandle)

'Check return code
```

```
  If RetCode <> RBX_E_SUCCESS Then
    DisplayRobixError "Error restarting robot.", RetCode
    Exit Sub
  End If

End Sub



'*************************************************************************
'This error-reporting function accepts a variable number of items and
'displays them in a message box along with an OK button. The user must
'click OK to continue.
'*************************************************************************
Public Sub DisplayError(ParamArray Items())


  'Loop through each item and concatenate it to the message string
  For Each X In Items
    Msg = Msg & X
  Next X

  'Display error message
 MsgBox Msg, 0, "Robix Error"
 End Sub

'*************************************************************************
'This error-reporting function accepts a error message string and a Robix
'error code. The description of the Robix error code is obtained, and
'the error message, the error code, and the error description are then
'displayed in a message box.
'*************************************************************************
Public Sub DisplayRobixError(ErrMsg As String, ErrCode As Long)
  On Error GoTo FnError


  'Get the size of the error description so we know how big our character
  'array will need to be in order to hold the description. Note return code.
  RetCode = rbxErrMsgSizeOf(ErrCode, DescripSiz)
  If RetCode <> RBX_E_SUCCESS Then
    DisplayRobixError "Error getting size of error description.", RetCode
    Exit Sub
  End If
```

```
  'Make sure RobotName is large enough to hold the robot's name by filling it
  'with the required number of characters. (DescripSiz was acquired by calling
  'rbxRobotNameSizeOf() above.) Note that any character can be used
  'here; we just use zero for simplicity.
  ErrDescrip = String(DescripSiz, "0")

  'Get the error description. Note return code.
  RetCode = rbxErrMsgGet(ErrCode, ErrDescrip, DescripSiz, RetSiz)
  If RetCode <> RBX_E_SUCCESS Then
    DisplayRobixError "Error getting error description.", RetCode
    Exit Sub
  End If

  'Display the entire error message
  'Note that we display the Robix error code according to the guidelines
  'discussed in RascalDLL.bas
  DisplayError ErrMsg & Chr(10) & "Robix Error Code: RBX_" & ErrCode & Chr(10)
& _
          "Error Description: " & ErrDescrip

  Exit Sub

FnError:
  'Display the error message (description unavailable)
  DisplayError ErrMsg & Chr(10) & "Robix Error Code: RBX_" & ErrCode & Chr(10)
& _
          "Error Description: unavailable"

End Sub



'*************************************************************************
'This function illustrates how to obtain a fresh switch reading safely.
'Ei switch values are read by the DLL only on demand, i.e. only when
'requested by an application. The function returns -1 if the switch is not
'read successfully.
'*************************************************************************
Public Function SwitchParkAndRead(EiHand As Long, SwitchID As Long) As
Long


  'Initialize time variables. Note that "Time()" is a VB function
  InitialTime = Time()
  totaltime = 0
```

```
'************************
'*** Read initial mod ***
'************************

'Read mod. This signals DLL to obtain a fresh reading. If this
'function returns RBX_E_SUCCESS, FirstMod will contain the requested
'switch mod. Note return code.
RetCode = rbxSwitchModGet(EiHand, SwitchID, FirstMod)

'Check return code
If RetCode <> RBX_E_SUCCESS Then
  DisplayRobixError "Error reading mod for switch " & SwitchID & ".", RetCode
  SwitchParkAndRead = -1
  Exit Function
End If

'This while loop reads the current switch mod ("modification count") repeatedly until the
'current mod differs from the first mod that was read.  When the
'two differ, this indicates that the switch value has been refreshed ("modified").
'Also within this while loop, we check to see if the Ei is responding
'properly, and we do a check to make sure that this loop is not executed
'indefinitely.

'Note to advanced programmers:
'Readings from the Ei's switch and analog inputs can take up to about 20 milliseconds.
'For many applications this delay is not a problem, and the program can simply
'wait for a new reading when desired. That is what we do below.
'On the other hand, for applications where several different inputs
'need to be read continuously, the waiting time may become excessive
'and the programmer may need to devise appropriate techniques to maintain
'a continuous set of readings with little total waiting overhead.
'This is left as an exercise to the advanced programmer.

Do
  '************************
  '*** Check Ei status ***
  '************************

  'Check status of Ei. If this function returns RBX_E_SUCCESS, EiOK
  'will contain the status of the Ei. Note return code.
  RetCode = rbxLptEiResponding(EiOK)

  'Check return code
  If RetCode <> RBX_E_SUCCESS Then
```

```
    DisplayRobixError "Error checking status of Ei.", RetCode
    SwitchParkAndRead = -1
    Exit Function
  End If

  'If Ei is not responding, show message.
  If EiOK = 0 Then
    'Create message
    ErrMsg = "Electronics Interface not responding. "
    ErrMsg = ErrMsg & "Check cables and power light."

    'Display message in a message box with buttons Retry and Cancel
    RetryCancel = MsgBox(ErrMsg, vbRetryCancel, "Robix Error")

    'If Cancel was clicked, exit this function. Otherwise,
    'continue trying to read switch mod.
    If RetryCancel = vbCancel Then
      SwitchParkAndRead = -1
      Exit Function
    End If
  End If

  '**********************
  '*** Check max time ***
  '**********************

  'Calculate total time parked
  totaltime = Time() - InitialTime

   'Check for max timeout period elapsed. This is done to prevent the
   'program from hanging if the switch mod doesn't change.
   'Note that RBX_MAX_EI_TIMEOUT is converted from milliseconds to
   'seconds for comparison with TotalTime
  If totaltime > RBX_MAX_EI_TIMEOUT / 1000 Then
    'Create message
    ErrMsg = "Unknown error reading switch " & SwitchID & ". "
    ErrMsg = ErrMsg + "Switch mod not updated within maximum timeout period."

    'Display message in a message box with buttons Retry and Cancel
    RetryCancel = MsgBox(ErrMsg, vbRetryCancel, "Robix Error")

    'If Retry was clicked, reset InitialTime. If Cancel was clicked,
    'exit this function.
    If RetryCancel = vbRetry Then
      InitialTime = Time()
    Else
```

```
        SwitchParkAndRead = -1
        Exit Function
      End If
    End If

    '*************************
    '*** Read current mod ***
    '*************************

    'Now we read current mod. Note return code.
    RetCode = rbxSwitchModGet(EiHand, SwitchID, CurrMod)

    'Check return code
    If RetCode <> RBX_E_SUCCESS Then
      DisplayRobixError "Error reading mod for switch " & SwitchID & ".", RetCode
      SwitchParkAndRead = -1
      Exit Function
    End If

    'Allow other applications to process their events
    DoEvents
  Loop While CurrMod = FirstMod

  'We get here when the current mod is different from the first mod.
  'This indicates that the switch's value has been refreshed.

  'Read fresh switch value. If this function returns RBX_E_SUCCESS,
  'SwitchValue will contain the current switch value. Note return code.
  RetCode = rbxSwitchValueGet(EiHand, SwitchID, SwitchValue)

  'Check return code
  If RetCode <> RBX_E_SUCCESS Then
    DisplayRobixError "Error reading value for switch " & SwitchID & ".", RetCode
    SwitchParkAndRead = -1
    Exit Function
  End If

  'Return switch value
  SwitchParkAndRead = SwitchValue
```

End Function

```
'*************************************************************************
'This function illustrates how to obtain a fresh Adc reading safely.
'Ei Adc values are read by the DLL only on demand, i.e. only when
'requested by an application. The function returns -1 if the Adc is not
'read successfully.
'*************************************************************************

Public Function AdcParkAndRead(EiHand As Long, AdcID As Long)


  'Initialize time variables. Note that "Time()" is a VB function
  InitialTime = Time()
  totaltime = 0

  '************************
  '*** Read initial mod ***
  '************************

  'Read mod. This signals DLL to obtain a fresh reading. If this
  'function returns RBX_E_SUCCESS, FirstMod will contain the requested
  'Adc mod. Note return code.
  RetCode = rbxAdcModGet(EiHand, AdcID, FirstMod)

  'Check return code
  If RetCode <> RBX_E_SUCCESS Then
    DisplayRobixError "Error reading mod for Adc " & AdcID & ".", RetCode
    AdcParkAndRead = -1
    Exit Function
  End If

  'This while loop reads the current Adc mod ("modification count") repeatedly until the
  'current mod differs from the first mod that was read.  When the
  'two differ, this indicates that the Adc value has been refreshed ("modified").
  'Also within this while loop, we check to see if the Ei is responding
  'properly, and we do a check to make sure that this loop is not executed
  'indefinitely.

  'Note to advanced programmers:
  'Readings from the Ei's Adc and analog inputs can take up to about 20 milliseconds.
  'For many applications this delay is not a problem, and the program can simply
  'wait for a new reading when desired. That is what we do below.
  'On the other hand, for applications where several different inputs
  'need to be read continuously, the waiting time may become excessive
  'and the programmer may need to devise appropriate techniques to maintain
  'a continuous set of readings with little total waiting overhead.
  'This is left as an exercise to the advanced programmer.
```

```
Do
  '**********************
  '*** Check Ei status ***
  '**********************

  'Check status of Ei. If this function returns RBX_E_SUCCESS, EiOK
  'will contain the status of the Ei. Note return code.
  RetCode = rbxLptEiResponding(EiOK)

  'Check return code
  If RetCode <> RBX_E_SUCCESS Then
    DisplayRobixError "Error checking status of Ei.", RetCode
    AdcParkAndRead = -1
    Exit Function
  End If

  'If Ei is not responding, show message.
  If EiOK = 0 Then
    'Create message
    ErrMsg = "Electronics Interface not responding. "
    ErrMsg = ErrMsg & "Check cables and power light."

    'Display message in a message box with buttons Retry and Cancel
    RetryCancel = MsgBox(ErrMsg, vbRetryCancel, "Robix Error")

    'If Cancel was clicked, exit this function. Otherwise,
    'continue trying to read Adc mod.
    If RetryCancel = vbCancel Then
      AdcParkAndRead = -1
      Exit Function
    End If
  End If

  '**********************
  '*** Check max time ***
  '**********************

  'Calculate total time parked
  totaltime = Time() - InitialTime

   'Check for max timeout period elapsed. This is done to prevent the
   'program from hanging if the Adc mod doesn't change.
   'Note that RBX_MAX_EI_TIMEOUT is converted from milliseconds to
   'seconds for comparison with TotalTime
  If totaltime > RBX_MAX_EI_TIMEOUT / 1000 Then
```

```
      'Create message
      ErrMsg = "Unknown error reading Adc " & AdcID & ". "
      ErrMsg = ErrMsg + "Adc mod not updated within maximum timeout period."

      'Display message in a message box with buttons Retry and Cancel
      RetryCancel = MsgBox(ErrMsg, vbRetryCancel, "Robix Error")

      'If Retry was clicked, reset InitialTime. If Cancel was clicked,
      'exit this function.
      If RetryCancel = vbRetry Then
        InitialTime = Time()
      Else
        AdcParkAndRead = -1
        Exit Function
      End If
    End If

    '*************************
    '*** Read current mod ***
    '*************************

    'Now we read current mod. Note return code.
    RetCode = rbxAdcModGet(EiHand, AdcID, CurrMod)

    'Check return code
    If RetCode <> RBX_E_SUCCESS Then
      DisplayRobixError "Error reading mod for Adc " & AdcID & ".", RetCode
      AdcParkAndRead = -1
      Exit Function
    End If

    'Allow other applications to process their events
    DoEvents
  Loop While CurrMod = FirstMod

  'We get here when the current mod is different from the first mod.
  'This indicates that the Adc's value has been refreshed.

  'Read fresh Adc value. If this function returns RBX_E_SUCCESS,
  'AdcValue will contain the current Adc value. Note return code.
  RetCode = rbxAdcValueGet(EiHand, AdcID, AdcValue)

  'Check return code
  If RetCode <> RBX_E_SUCCESS Then
    DisplayRobixError "Error reading value for Adc " & AdcID & ".", RetCode
    AdcParkAndRead = -1
```

```
    Exit Function
  End If

  'Return Adc value
  AdcParkAndRead = AdcValue

End Function


Private Sub Timer1_Timer()

'tests if we started the application
If ProgramRunning = 1 Then

MatlabReady = XLApp.Cells(18, 5)

   'tests if Matlab has been launched, so if everything is ready
   If MatlabReady = 1 Then

      'tests if the robot is waiting for instruction. This is to avoid problem
      'due to the fact the robot is already executing a command
      If rbxRobotAction_Wait Then

         'Tell Matlab to pause calculations
         NextCalculation = 0
         XLApp.Cells(18, 7) = NextCalculation



'*********************************************************************
         'Retrieve the new joints increments on the Excel sheet

'*********************************************************************

         strMoveJ1 = XLApp.Cells(18, 2)
         strMoveJ2 = XLApp.Cells(18, 3)
```

```
'*********************************************************************
        'Sends the move command to the robot

'*********************************************************************

        'Write the script to be executed
        Script = "Move 1 to " + strMoveJ1 + ", 2 to " + strMoveJ2

        'Execute the script
        RetCode = rbxScriptExecute(RobotHandle, Script)

        'waits for the robot to be done with the string execution
        Do
           'Allow other applications to process their events
           DoEvents
        Loop Until rbxRobotAction_Wait

        'realizes a pause so the camera has time to update the coordinates
        Dim pausetime, Start

        'Make a pause to be sure Framework has time enough to update coordinates
        pausetime = 0.2
        Start = Timer
        Do While Timer < Start + pausetime
           'Allow other applications to process their events
           DoEvents
        Loop

        'Tell Matlab to make a new calculation
        NextCalculation = 1
        XLApp.Cells(18, 7) = NextCalculation

      Else
        'Allow other applications to process their events
        DoEvents
      End If

   End If

End If

End Sub
```

Appendix 3

Introduction to TCP/IP

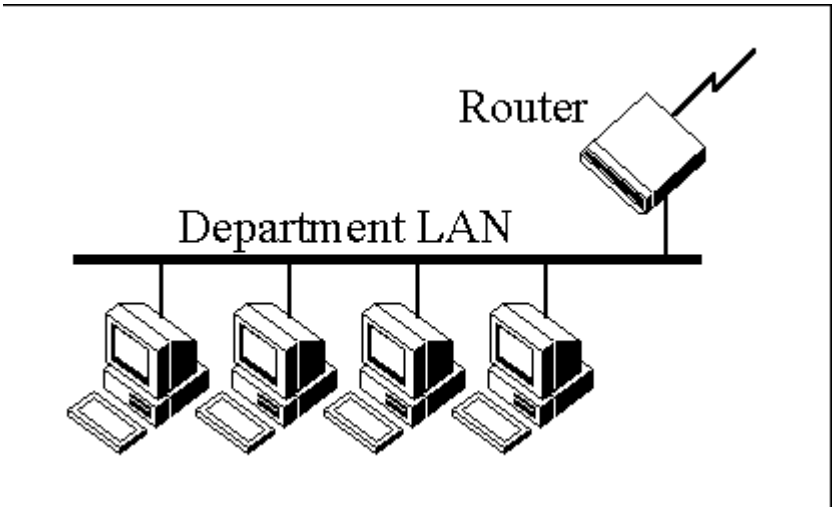
TCP and IP were developed by a Department of Defense (DOD) research project to connect a number different networks designed by different vendors into a network of networks (the "Internet"). It was initially successful because it delivered a few basic services that everyone needs (file transfer, electronic mail, remote logon) across a very large number of client and server systems. Several computers in a small department can use TCP/IP (along with other protocols) on a single LAN. The IP component provides routing from the department to the enterprise network, then to regional networks, and finally to the global Internet. On the battlefield a communications network will sustain damage, so the DOD designed TCP/IP to be robust and automatically recover from any node or phone line failure. This design allows the construction of very large networks with less central management. However, because of the automatic recovery, network problems can go undiagnosed and uncorrected for long periods of time.

As with all other communications protocol, TCP/IP is composed of layers:

- **IP** - is responsible for moving packet of data from node to node. IP forwards each packet based on a four byte destination address (the IP number). The Internet authorities assign ranges of numbers to different organizations. The organizations assign groups of their numbers to departments. IP operates on gateway machines that move data from department to organization to region and then around the world.

- **TCP** - is responsible for verifying the correct delivery of data from client to server. Data can be lost in the intermediate network. TCP adds support to detect errors or lost data and to trigger retransmission until the data is correctly and completely received.

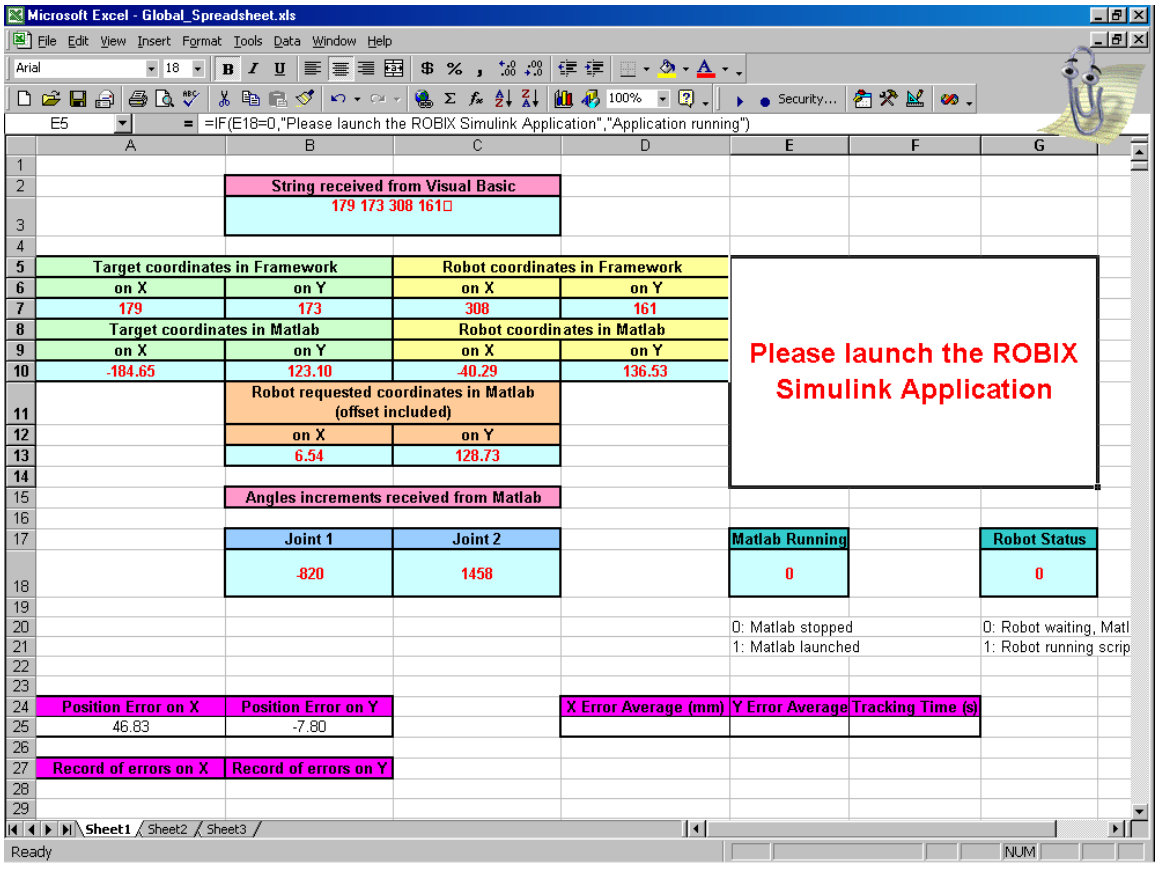
- **Sockets -** is a name given to the package of subroutines that provide access to TCP/IP on most systems.



Appendix 4

Excel Platform

This Excel spreadsheet has been created to realize the link between Matlab and Visual Basic. Its existence is due to the ease of use when it is linked with both of these sofwares.



Appendix 5

QUASI-NEWTON METHOD

Here is the quasi-Newton algorithm. Its explanation is well described in Jenelle Piepmeier thesis. Please refer to it for further indications.

Algorithm (Dynamic Quasi-Newton Method with RLS estimation) *Given*

$f : \mathbb{R}^n \rightarrow \mathbb{R}^m; \theta_0, \theta_1 \in \mathbb{R}^n; \hat{J}_0 \in \mathbb{R}^{m \times n}, P_0 \in \mathbb{R}^{n \times n}, \lambda \in (0,1)$

Do for $k = 1, 2, ...$

$$\Delta f = f_k - f_{k-1}$$

$$h_\theta = \theta_k - \theta_{k-1}$$

$$\hat{J}_k = \hat{J}_{k-1} + \left(\lambda + h_\theta^T P_{k-1} h_\theta\right)^{-1} \left(\Delta f - \hat{J}_{k-1} h_\theta + \frac{\partial y_k(i)}{\partial i} h_i\right) h_\theta^T P_{k-1}$$

$$P_k = \frac{1}{\lambda}\left(P_{k-1} - \left(\lambda + h_\theta^T P_{k-1} h_\theta\right)^{-1} \left(P_{k-1} h_\theta^T h_\theta P_{k-1}\right)\right)$$

$$\theta_{k+1} = \theta_k - \left(\hat{J}_k^T \hat{J}_k\right)^{-1} \hat{J}_k^T \left(f_k - \frac{\partial y_k(i)}{\partial i} h_i\right)$$

End for

With :

- $(.)_k$ denotes Kth iteration
- λ forgetting factor
- θ a joint angle
- f the residual error in the sensor space
- J_k the Jacobian at iteration k

Appendix 6

MATLAB M-FILES

Following are some interesting M-Files linked to our main Simulink application.

%%%
%
% AdeptFKProc.m
% This function uses the foward kinematics of the adept robot
% to determine the position of the end-effector in XYZ coordinates
% from the joint angles of the robot. Z coordinates will be ignored.
% INPUT: joint angles of adept
% OUTPUT: xy coordinates of end-effector
%
%%%

```
function out=main(inJ)                    % inJ - joint values of adept
% declare global variables used
global J3
global J4                                 % constant value given for joint 4 of adept
global J5                                 % constant value given for joint 5 of adept

inJ=[inJ;J3;J4;J5];                       % expand joint vector to include last three joint values

T1=DHP(0.0,0.0,0,inJ(1));                 % these transormations from the internet
T2=DHP(0.0,93.28,0,inJ(2));               % (adept web site) but with all z
T3=DHP(0.0,95.71,-inJ(3),0);              % components taken out (i.e. in T1, T6)
T4=Rz(-inJ(4));                           %
T5=Ry(-inJ(5));                           %
T6=Translate([0;0;0]);                    % To offset to flat mounting plate
T=T1*T2*T3*T4*T5*T6;                      % determine total transformation matrix

T=T(1:2,4);
```

```
out=T;

%*** Now update the camera transformation ***********************************

global Tcam
T_Total = T1*T2*T3;
Tcam = inv(T_Total);

%*******************************************************************************
% Translation function

function T=Translate(V)
T=eye(4,4)+[zeros(4,3) [V;0]];

%*******************************************************************************
% These functions are rotation matrices about the Y and Z axes respectively
%       They are used above in finding the Adept transformation matrices.

function T=Rx(inTheta)
inTheta=inTheta*pi/180;
T=[1, 0, 0, 0,
  0, cos(inTheta), -sin(inTheta), 0,
  0, sin(inTheta), cos(inTheta), 0,
  0, 0, 0, 1];

function T=Ry(inTheta)
inTheta=inTheta*pi/180;
T=[cos(inTheta), 0, sin(inTheta), 0
  0, 1, 0, 0
  -sin(inTheta), 0, cos(inTheta), 0
  0, 0, 0, 1];

function T=Rz(inTheta)
inTheta=inTheta*pi/180;
T=[cos(inTheta), -sin(inTheta), 0, 0,
  sin(inTheta), cos(inTheta), 0, 0,
  0, 0, 1, 0,
  0, 0, 0, 1];
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% AnglesOutput.m
%       This function sends new joint angles onto the Excel Spreadsheet
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [A,B]= (sys)

   A=sys(1,1)
   B=sys(2,1)

   % initiate a conversation with Excel from spreadsheet "Global_Spreadsheet.xls"
   channel = ddeinit ('excel', 'Global_Spreadsheet.xls');

   % Set ranges of cells in Excel for poking.
   range1 = 'r20c2';
   range2 = 'r20c3';

   % enter the datas we want to export
   a = [A];
   b = [B];

   % send a matrix  to Excel
   rc = ddepoke(channel, range1,a );
   rc = ddepoke(channel, range2,b );

   % terminate the communication with Excel
   rc = ddeterm (channel);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% CameraPlotsPoints.m
%       This function takes the data in the DataOut file (a file of the
%               feature points for all time increments) and plots the target
feature
%               position and the end effector position together.
%
%       INPUT:  data file DataOut
%       OUTPUT: plot shown by double clicking the 'Cameras' button in
simulink
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function main

% call global variables to be used
global CamDataOut          % file of virtual target and end effector feature
positions

% First check to see if there is any data in the file (there will be no data
%       if the simulation has not been run) and give an error message for no data.
if isempty(CamDataOut)
   msgbox('Currently no data available.','Error');
   return;
end

[n,m] = size(CamDataOut);
target_x_data = [];
target_y_data = [];
robot_x_data = [];
robot_y_data = [];

for i = 10:30:n
   target_x_data = [target_x_data;CamDataOut(i,1)];
   target_y_data = [target_y_data;CamDataOut(i,2)];
   robot_x_data = [robot_x_data;CamDataOut(i,3)];
   robot_y_data = [robot_y_data;CamDataOut(i,4)];
end

[p,q] = size(target_x_data);
```

```
% If there is data, then the plots can be created:
% Create a plot window
figure('Units','normalized','Position',[.1 .1 .8 .8]);
% Plot Data
hold on
plot(target_x_data(1:p),target_y_data(1:p),'+',robot_x_data(1:p),robot_y_data(1:p
),'o');
title('Target and Robot Positioning in Tracking','fontweight','bold','fontsize',14);
xlabel('X','fontweight','bold','fontsize',14)
ylabel('Y','fontweight','bold','fontsize',14)
legend('target position','robot position')
axis([-10 10 -10 10])
% Ensure true size
daspect([1 1 1]);




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%       SimInit.m
%         This program initializes neccessary global variables and files
%
% This function is also called inside cameramodel.m to reinitialize
% without having to run this file everytime.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global DataOut                          % XY feature coordinate data
used for plotting
DataOut=[];
global rmserror                         % XY feature coordinate data
used for plotting
rmserror=[];

% Frame Timing
global FT_TOTAL                 % Total time of a complete update cycle
global FT_PROCESSING            % Time Required for image processing
and controls
global FT_COMMUNICATION         % Time required for commands to get to
the adept


FT_PROCESSING=11/1000;
```

```
FT_COMMUNICATION=1/1000;
FT_TOTAL=25/1000;

% Adept Robot
global AR_JOINTCOUNT                % number of joints in the robot
global AR_JOINTSTART                % initial joint positions
global JointsK                      % joint values at time increment k
global JointsK1                     % joint values at time increment k-1
global MaxJointRates                % maximum joint velocity
global dJoints                      % desired joint changes
global dfdt                         % calculated velocity of target
global fk                           % error values at time increment k
global fk1                          % error values at time increment k-1
dJoints = [0;0];
dfdt=[0;0];
fk = [0;0];

% Joints 3, 4 and 5 are hardwired for now only a 3 DOF problem
% IMPORTANT if you change these reinitialize manually
global J3
global J4
global J5
J3=0;
J4=0;
J5=0;

% RLS Controller
global lambda               % 'memory' or forgetting factor for Jacobian update
lambda = 1;

AR_JOINTCOUNT=2;

% Joint Initialization
% IMPORTANT: If you change this value you must double click
% on initialize because this file is normally called by CameraModel
% which is after MATLAB initilizes the blocks
%AR_JOINTSTART=[-100;100];
AR_JOINTSTART=[0;0];

%MaxJointRates=[7 7]';
%MaxJointRates=[70 70]';
MaxJointRates=[500 500]';

global JK                   % the Jacobian at time increment k
global PK                   % the P matrix at time increment K used in finding
new Jacobian
```

```
JK = eye(AR_JOINTCOUNT);
PK = eye(AR_JOINTCOUNT);                         % initialization of P
matrix

% Initialize Joints
JointsK=AR_JOINTSTART;               % set joints to initial values
JointsK1=JointsK;                % give the k-1 time increment joint values also

% Initialize MaxJoint values
global MaxJoint
MaxJoint = 0;

% Initialize target path counter used in some target paths
global path_counter
path_counter = 0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% RobotInput.m
%     This function retrieves robot coordinates from the Excel
%Spreadsheet
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function P= (W)

   X=W+1;

   % initiate a conversation with Excel from spreadsheet
"Global_Spreadsheet.xls"
   channel = ddeinit ('excel', 'Global_Spreadsheet.xls');

   % Set cells in Excel for poking
   C10 = 'r10c3';
   D10 = 'r10c4';

   % request the values from the excel spreadsheet
   RobotX = ddereq (channel,C10);
   RobotY = ddereq (channel,D10);
   P=[
      RobotX
      RobotY
   ];
    % terminate the communication with Excel
   rc = ddeterm (channel);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% TargetPath.m
%       This function produces a time dependant path for the target position
%         in the XYZ robot coordinates.
%
%       INPUT:  clock time
%       OUTPUT: target position, as vector P
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%




function P=TargetPath(inTime)               % inTime - time from clock

%Pick the Path to be followed by the target:
% Path 1:     A circle of specified radius
% Path 2:     Linearly back and forth in the x-direction, slowing at the ends
% Path 3:     Linearly back and forth, no slowing
% Path 4:     A square, slowing at the corners
% Path 5:     A square, no slowing at the corners
% Path 6:     A Sun Shape
% Path 7:     Curly Shape
path = 1;
global path_counter

% the target will travel in a circle w.r.t. the x and y axes, in the z=0 plane
if path == 1
        spc = 50;
        r=inTime*(2*pi/spc); % control speed of target around path (spc - sec. per
cycle)

        P = [
        -40*cos(r) - 305
        40*sin(r) + 450
  ];
end

% target travels back and forth in a line in the x direction, slowing at ends

if path == 2
  spc = 40;
  r=inTime*(2*pi/spc);
```

```
  P = [
        -100*cos(r)
        0
  ];
end

% target travels back and forth in x-direction, no slowing at ends
if path == 3
  spc = 10;
  n = 0;
  timer = inTime - 2*n*spc;
  if timer<spc
    X_position = (timer)*(175/spc);          % 175 - length of line
  else
    X_position = 175 - (timer-spc)*(175/spc);
    if timer>spc
      n = n + 1;
    end
  end
  P = [
        X_position - 87.5                % subtract 87.5 center the line
        0
  ];
end

% target travels in square, slowing at corners

if path == 4
  spc = 10;
  timer = inTime - spc*(path_counter)*4;
  if timer<spc
    r=timer*(pi/spc);
    X_position = -87.5*cos(r);
    Y_position = -87.5;
  elseif timer<2*spc
    r=timer*(pi/spc);
    X_position = 87.5;
    Y_position = 87.5*cos(r);
  elseif timer<3*spc
    r=timer*(pi/spc);
    X_position = 87.5*cos(r);
    Y_position = 87.5;
  else
    r=timer*(pi/spc);
    X_position = -87.5;
    Y_position = -87.5*cos(r);
```

```
    if round(1000*timer) == 4000*spc
      path_counter = path_counter+1;
    end
  end
  P = [
       X_position;
       Y_position;
  ];
end

% target travels in square with no slowing at corners
if path == 5
  spc = 10;
  timer = inTime - spc*(path_counter)*4;
  if timer<spc
    t = inTime - spc*(path_counter)*4;
    X_position = t*(175/spc);
    Y_position = 0;
  elseif timer<2*spc
    t = inTime - spc*((path_counter)*4 + 1);
    X_position = 175;
    Y_position = t*(175/spc);
  elseif timer<3*spc
    t = inTime - spc*((path_counter)*4 + 2);
    X_position = 175 - t*(175/spc);
    Y_position = 175;
  else
              t = inTime - spc*((path_counter)*4 + 3);
    X_position = 0;
    Y_position = 175 - t*(175/spc);
    if round(1000*t) == 1000*spc
      path_counter = path_counter+1;
    end
  end
  P = [
       X_position - 87.5;
       Y_position - 87.5;
  ];
end

% Fireball
if path == 6
  spc1 = 40;
  spc2 = spc1/6;
  r1 = inTime*(2*pi/spc1);
  r2 = inTime*(2*pi/spc2);
```

```
        P = [
    (-100 - 20*cos(r2))*cos(r1+cos(r2)/5)
                (100 + 20*cos(r2))*sin(r1+cos(r2)/5)
  ];
end

% Curly
if path == 7
  spc1 = 25;
  spc2 = spc1/8;
  r1 = inTime*(2*pi/spc1);
  r2 = inTime*(2*pi/spc2);
        P = [
    (-100 - 20*cos(r2))*cos(r1-sin(r2)/4)
                (100 + 20*cos(r2))*sin(r1-sin(r2)/4)
  ];
end

Adept=AdeptFKProc([0;-120]);     % place target in adept workspace
P=P+Adept;
%P = Adept;          % use to determine circle center location
```

Appendix 7

USER MANUAL

To use our application, you will have to follow these steps :

1. First, check all your PC Software. On your machine should be installed : Framework 2.1, Visual Basic 6.0 and Matlab 6.0 and Rascal (Robix appl.)

2. Plug the camera correctly according to its manual on your PC ethernet card.

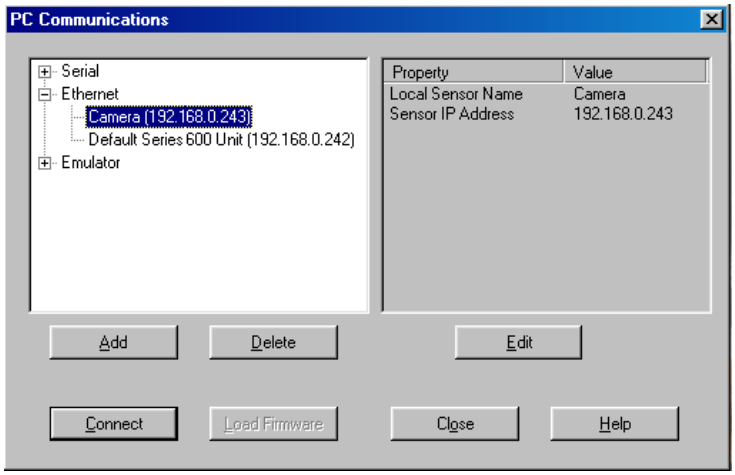
3. Check your configuration. Your ethernet card must have the following IP address : 192.168.0.240 , Submask 255.255.255.0

4. Check your progam paths. Within « Robot Control.vbp », you may change paths to fit your settings parameters. One is in the Form_Load() sub and the other in the Start_Click() sub.
   If changes are necessary, modify the paths according to your system, save the project and make Visual Servoing.exe by choosing this option in the File menu.

You are now ready to use our application on your PC.

5. Now, launch Framework 2.1 by clicking on the appropriate icon. 

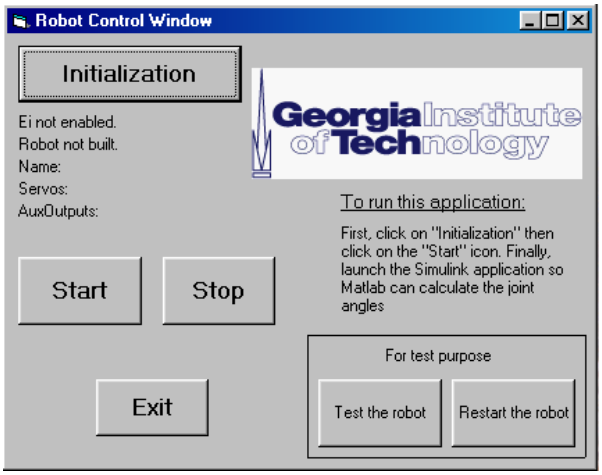
6. Choose Ethernet, click on « Camera (192.168.0.243) » and click Connect



7. Now, select the « Robot Control » product on the Framework menu and click Start. Minimize the window.
8. Launch « Visual Servoing.exe » by clicking on the icon : 

9. Click « Initialization ». For test purpose, you may want to click « Test the Robot » and verify the robot moved its both joints. Once the test is done, don't forget to click « Restart the Robot ».

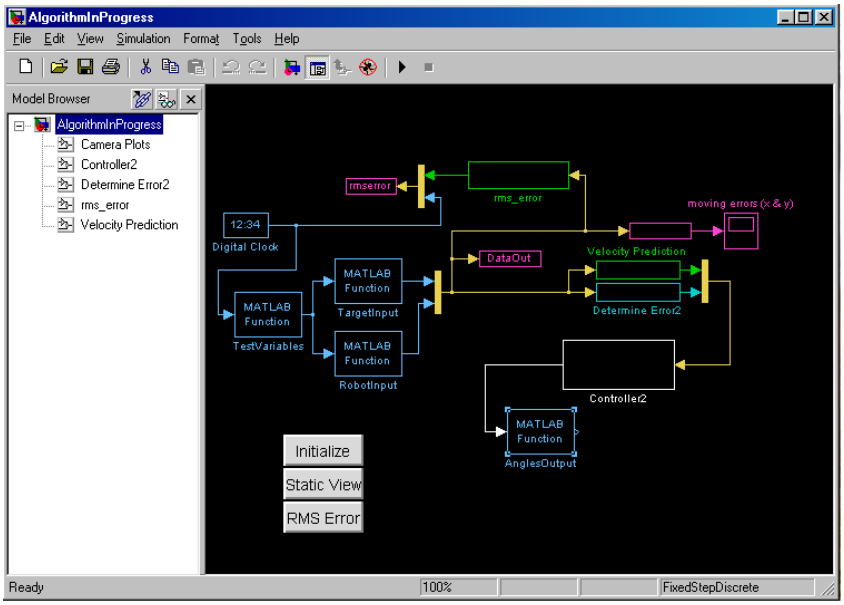
10. When you are ready to launch the application, click on the Start button.



11. Launch Matlab by clicking its icon 

. Open the Simulink file called Algorithm.

12. Click Initialize and then Start. The application is now running.



You may stop the application anytime by clicking on Stop or by closing windows.

This report has been written with MS Word 2000
Achieved and printed on June the 15th 2001