Eksperimentu stenda lietotāja saskarnes galvenā loga vadības loģikas kods

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.IO;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;
using UR10.Patterns;
using static UR10.PatternDriver;

namespace UR10
{
   public partial class FormMain : Form, IUr10Out, IPatternDriverOut, ITcpServerOut
   {
      private IConnection connection = new Connection();
      private IPatternDriver patternDriver;
      private ISpiralBuilder spiralBuilder = new SpiralBuilder();
      private IZigZagBuilder zigZagBuilder = new ZigZagBuilder();
      private IUr10StreamDecoder decoder;
      private ITcpServer tcpServer;

      private double headXCoodrinateInPixels = 0;
      private double headYCoordinateInPixels = 0;

      private float k = 3;
      private float scale = 10;
      private int baseEllipseSize = 20;
      private int headPointerSize = 6;

      private Rectangle box1;
      private Rectangle box2;

      private List<PointF[]> executedPatternCurves = new List<PointF[]>();

      public FormMain()
      {
         InitializeComponent();

         box1 = new Rectangle((int)(50 / k), (int)(990 / k), (int)((float)Properties.Settings.Default.BoxLength / k), (int)(150 / k));
         box2 = new Rectangle((int)(50 / k), (int)(720 / k), (int)((float)Properties.Settings.Default.BoxLength / k), (int)(150 / k));

         try
         {
            checkBoxEnableRow1.Checked = Properties.Settings.Default.Row1Enabled;
            checkBoxEnableRow2.Checked = Properties.Settings.Default.Row2Enabled;
            checkBoxEnableRow3.Checked = Properties.Settings.Default.Row3Enabled;
            checkBoxEnableRow4.Checked = Properties.Settings.Default.Row4Enabled;

            numericUpDownRow1Distance.Value = Properties.Settings.Default.Row1Distance;
            numericUpDownRow2Distance.Value = Properties.Settings.Default.Row2Distance;
            numericUpDownRow3Distance.Value = Properties.Settings.Default.Row3Distance;
            numericUpDownRow4Distance.Value = Properties.Settings.Default.Row4Distance;

            numericUpDownWeedColumns.Value = Properties.Settings.Default.WeedColumns;
            numericUpDownWeedInterval.Value = Properties.Settings.Default.WeedInterval;

            if(Directory.Exists(Properties.Settings.Default.PhotosFolderPath))
            {
               folderBrowserDialog.SelectedPath = Properties.Settings.Default.PhotosFolderPath;
            }
```

```
            else
            {
                if(DialogResult.OK == folderBrowserDialog.ShowDialog())
                {
                    Properties.Settings.Default.PhotosFolderPath = folderBrowserDialog.SelectedPath;
                    Properties.Settings.Default.Save();
                }
            }
        }
        catch (Exception exc)
        {
            MessageBox.Show(exc.ToString(), "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }

        labelScale.Text = $"Scale 1:{scale}";
    }

    public void CommandFinished()
    {
        if (patternDriver != null)
        {
            patternDriver.CommandFinished();
        }
    }

    public void PatternCompleted()
    {
        SetControlModeButtonsState(true);
    }

    public void EmergencyStopStatusReceived(bool status)
    {
        if (patternDriver != null)
        {
            patternDriver.EmergensyStopStatusReceived(status);
        }
    }

    public void SendLogMessage(string message)
    {
        LogMessage(message);
    }

    public void CoordinatesReceived(double x, double y, double z, double rX, double rY, double rZ)
    {
        headXCoodrinateInPixels = (-x + (double)Properties.Settings.Default.BaseXCoordinate) / k;
        headYCoordinateInPixels = (-y + (double)Properties.Settings.Default.BaseYCoordinate) / k;

        if(patternDriver != null)
        {
            executedPatternCurves = patternDriver.GetExecutedPatterns().Select(p => p.Select(point => new
PointF(pictureBoxUrPlatform.Width - (-point.X * 1000 + (float)Properties.Settings.Default.BaseXCoordinate) / k, (-point.Y * 1000 +
(float)Properties.Settings.Default.BaseYCoordinate) / k)).ToArray()).ToList();
            patternDriver.UpdateCartesianData((float)x / 1000, (float)y / 1000, (float)z / 1000, (float)rX, (float)rY, (float)rZ);
        }

        Invoke((MethodInvoker)delegate
            {
                labelX.Text = $"X = {Math.Round(x, 2)}mm";
                labelY.Text = $"Y = {Math.Round(y, 2)}mm";
                labelZ.Text = $"Z = {Math.Round(z, 2)}mm";
                pictureBoxUrPlatform.Refresh();
            }
        );
    }

    private void tabControl_Selecting(object sender, TabControlCancelEventArgs e)
    {
        if(e.TabPage.Text == "Admin")
        {
```

```
        LogMessage("Admin panel access attempt");
        FormAdminPasswordPrompt formAdminPasswordPrompt = new FormAdminPasswordPrompt();
        e.Cancel = DialogResult.OK != formAdminPasswordPrompt.ShowDialog();
        if(e.Cancel)
        {
            LogMessage("Admin panel access denied");
        }
        else
        {
            LogMessage("Admin panel access granted");
        }
    }
}

private void buttonConnect_Click(object sender, EventArgs e)
{
    LogMessage("Connecting to UR10. Please wait.");

    Task.Factory.StartNew(() =>
    {
        (bool isConnected, string exception) = connection.ConnectToUr10();

        if (isConnected)
        {
            LogMessage("Connected");

            LogMessage("Stoping controller program...");
            connection.SendServiceMessage("stop");
            LogMessage("Program stopped");

            LogMessage("Turning on robot...");
            connection.SendServiceMessage("power on");
            LogMessage("Power is on");

            LogMessage("Releasing brakes...");
            connection.SendServiceMessage("brake release");
            LogMessage("Brakes released");

            Stream stream = connection.GetStream();

            decoder = new Ur10StreamDecoder(this);
            Task.Factory.StartNew(() => decoder.StartDecoder(stream));

            tcpServer = new TcpServer(this);
            Task.Factory.StartNew(() => tcpServer.StartTcpServer());

            LogMessage("Starting controller program...");
            connection.SendServiceMessage("play");
            LogMessage("Program started");

            LogMessage("Waiting for TCP client");
            tcpServer.WaitForTcpClient();

            Invoke((MethodInvoker)delegate
            {
                buttonConnect.Enabled = !isConnected;
                buttonDisconnect.Enabled = isConnected;
                buttonRunPattern.Enabled = isConnected;
                buttonTransportationMode.Enabled = isConnected;
                buttonHomeMode.Enabled = isConnected;
                buttonStop.Enabled = isConnected;
            });

            LogMessage("TCP client connected");

            patternDriver = new PatternDriver(tcpServer, this);

            LogMessage("Turning digital and analog outputs off");
            patternDriver.TurnOffAll();
            patternDriver.SetPatternSpeed((float)Properties.Settings.Default.PatternSpeed / 1000);
```

```
        }
        else
        {
            LogMessage(exception);
        }
    });
}

private void buttonDisconnect_Click(object sender, EventArgs e)
{
    LogMessage("Disconnecting from UR10. Please wait.");

    Task.Factory.StartNew(() => tcpServer.StopTcpServer());

    Task.Factory.StartNew(() => decoder.StopDecoder());
    Thread.Sleep(1000);

    Task.Factory.StartNew(() =>
    {
        (bool isDisconnected, string exception) = connection.DisconnectFromUr10();

        Invoke((MethodInvoker)delegate
        {
            buttonConnect.Enabled = isDisconnected;
            buttonDisconnect.Enabled = !isDisconnected;
            buttonRunPattern.Enabled = !isDisconnected;
            buttonTransportationMode.Enabled = !isDisconnected;
            buttonHomeMode.Enabled = !isDisconnected;
            buttonStop.Enabled = !isDisconnected;
        });

        if (isDisconnected)
        {
            LogMessage("Disconnected");
        }
        else
        {
            LogMessage(exception);
        }
    });
}

private void LogMessage(string message)
{
    Invoke((MethodInvoker)delegate
          {
              listBoxConsole.Items.Insert(0, $"{DateTime.Now.ToLocalTime()}: {message}");
          }
    );
}

private void tabPageAdmin_Enter(object sender, EventArgs e)
{
    Task.Factory.StartNew(() => ReadSettings());
}

private void ReadSettings()
{
    try
    {
        Invoke((MethodInvoker)delegate
        {
            numericUpDownPatternStartAngle.Value = Properties.Settings.Default.PatternStartAngle;
            numericUpDownPatterDiameter.Value = Properties.Settings.Default.PatternDiameter;
            numericUpDownPatternStep.Value = Properties.Settings.Default.PatternStep;
            numericUpDownPatternSpeed.Value = Properties.Settings.Default.PatternSpeed;
            numericUpDownDeltaTheta.Value = Properties.Settings.Default.PatternDeltaTheta;
            numericUpDownBaseXCoordinate.Value = Properties.Settings.Default.BaseXCoordinate;
            numericUpDownBaseYCoordinate.Value = Properties.Settings.Default.BaseYCoordinate;
            numericUpDownTravelSpeed.Value = Properties.Settings.Default.TravelSpeed;
```

```
                textBoxIpAddress.Text = Properties.Settings.Default.IpAddress;
                numericUpDownWeedInterval.Value = Properties.Settings.Default.WeedInterval;
                comboBoxPatternType.SelectedIndex = Properties.Settings.Default.PatternType;
                checkBoxTakePhotos.Checked = Properties.Settings.Default.TakePhotos;
            });
        }
        catch(Exception exc)
        {
            MessageBox.Show(exc.ToString(), "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    private void numericUpDownPatterDiameter_ValueChanged(object sender, EventArgs e)
    {
        Properties.Settings.Default.PatternDiameter = numericUpDownPatterDiameter.Value;
        Properties.Settings.Default.Save();
        pictureBoxPatternSample.Refresh();
    }

    private void numericUpDownPatternStep_ValueChanged(object sender, EventArgs e)
    {
        Properties.Settings.Default.PatternStep = numericUpDownPatternStep.Value;
        Properties.Settings.Default.Save();
        pictureBoxPatternSample.Refresh();
    }

    private void numericUpDownPatternSpeed_ValueChanged(object sender, EventArgs e)
    {
        Properties.Settings.Default.PatternSpeed = numericUpDownPatternSpeed.Value;
        Properties.Settings.Default.Save();

        if(patternDriver != null)
        {
            patternDriver.SetPatternSpeed((float)numericUpDownPatternSpeed.Value / 1000);
        }
    }

    private void checkBoxEnableRow1_CheckedChanged(object sender, EventArgs e)
    {
        Properties.Settings.Default.Row1Enabled = checkBoxEnableRow1.Checked;
        Properties.Settings.Default.Save();

        pictureBoxUrPlatform.Refresh();
        SetPatternModeButtonsState(checkBoxEnableRow1.Checked || checkBoxEnableRow2.Checked ||
checkBoxEnableRow3.Checked || checkBoxEnableRow4.Checked || Properties.Settings.Default.TakePhotos);
    }

    private void checkBoxEnableRow2_CheckedChanged(object sender, EventArgs e)
    {
        Properties.Settings.Default.Row2Enabled = checkBoxEnableRow2.Checked;
        Properties.Settings.Default.Save();

        pictureBoxUrPlatform.Refresh();
        SetPatternModeButtonsState(checkBoxEnableRow1.Checked || checkBoxEnableRow2.Checked ||
checkBoxEnableRow3.Checked || checkBoxEnableRow4.Checked || Properties.Settings.Default.TakePhotos);
    }

    private void checkBoxEnableRow3_CheckedChanged(object sender, EventArgs e)
    {
        Properties.Settings.Default.Row3Enabled = checkBoxEnableRow3.Checked;
        Properties.Settings.Default.Save();

        pictureBoxUrPlatform.Refresh();
        SetPatternModeButtonsState(checkBoxEnableRow1.Checked || checkBoxEnableRow2.Checked ||
checkBoxEnableRow3.Checked || checkBoxEnableRow4.Checked || Properties.Settings.Default.TakePhotos);
    }

    private void checkBoxEnableRow4_CheckedChanged(object sender, EventArgs e)
    {
        Properties.Settings.Default.Row4Enabled = checkBoxEnableRow4.Checked;
```

```
            Properties.Settings.Default.Save();

            pictureBoxUrPlatform.Refresh();
            SetPatternModeButtonsState(checkBoxEnableRow1.Checked || checkBoxEnableRow2.Checked || checkBoxEnableRow3.Checked || checkBoxEnableRow4.Checked || Properties.Settings.Default.TakePhotos);
        }

        private void numericUpDownRow1Distance_ValueChanged(object sender, EventArgs e)
        {
            Properties.Settings.Default.Row1Distance = numericUpDownRow1Distance.Value;
            Properties.Settings.Default.Save();
        }

        private void numericUpDownRow2Distance_ValueChanged(object sender, EventArgs e)
        {
            Properties.Settings.Default.Row2Distance = numericUpDownRow2Distance.Value;
            Properties.Settings.Default.Save();
        }

        private void numericUpDownRow3Distance_ValueChanged(object sender, EventArgs e)
        {
            Properties.Settings.Default.Row3Distance = numericUpDownRow3Distance.Value;
            Properties.Settings.Default.Save();
        }

        private void numericUpDownRow4Distance_ValueChanged(object sender, EventArgs e)
        {
            Properties.Settings.Default.Row4Distance = numericUpDownRow4Distance.Value;
            Properties.Settings.Default.Save();
        }

        private void numericUpDownDeltaTheta_ValueChanged(object sender, EventArgs e)
        {
            Properties.Settings.Default.PatternDeltaTheta = numericUpDownDeltaTheta.Value;
            Properties.Settings.Default.Save();
            pictureBoxPatternSample.Refresh();
        }

        private void numericUpDownPatternStartAngle_ValueChanged(object sender, EventArgs e)
        {
            Properties.Settings.Default.PatternStartAngle = numericUpDownPatternStartAngle.Value;
            Properties.Settings.Default.Save();
            pictureBoxPatternSample.Refresh();
        }

        private void NumericUpDownBaseXCoordinate_ValueChanged(object sender, EventArgs e)
        {
            Properties.Settings.Default.BaseXCoordinate = numericUpDownBaseXCoordinate.Value;
            Properties.Settings.Default.Save();
        }

        private void NumericUpDownBaseYCoordinate_ValueChanged(object sender, EventArgs e)
        {
            Properties.Settings.Default.BaseYCoordinate = numericUpDownBaseYCoordinate.Value;
            Properties.Settings.Default.Save();
        }

        private void PictureBoxUrPlatform_Paint(object sender, PaintEventArgs e)
        {
            e.Graphics.Clear(pictureBoxUrPlatform.BackColor = Color.White);
            e.Graphics.SmoothingMode = SmoothingMode.AntiAlias;

            e.Graphics.FillEllipse(Brushes.SkyBlue, pictureBoxUrPlatform.Width - (float)Properties.Settings.Default.BaseXCoordinate / k - baseEllipseSize/2, (float)Properties.Settings.Default.BaseYCoordinate / k - baseEllipseSize/2, baseEllipseSize, baseEllipseSize);
            e.Graphics.FillEllipse(Brushes.Blue, pictureBoxUrPlatform.Width - (float)headXCoodrinateInPixels - headPointerSize/2, (float)(headYCoordinateInPixels - headPointerSize/2), headPointerSize, headPointerSize);

            for(int x = 0; x < executedPatternCurves.Count; x++)
            {
                if (executedPatternCurves[x].Length > 2)
```

```
                {
                    e.Graphics.DrawLines(Pens.Blue, executedPatternCurves[x]);
                }
            }

            if(Properties.Settings.Default.Row1Enabled || Properties.Settings.Default.Row2Enabled)
            {
                e.Graphics.DrawRectangle(Pens.OrangeRed, box1);
            }

            if (Properties.Settings.Default.Row3Enabled || Properties.Settings.Default.Row4Enabled)
            {
                e.Graphics.DrawRectangle(Pens.OrangeRed, box2);
            }
        }

        //Code from http://csharphelper.com/blog/2018/10/draw-an-archimedes-spiral-in-c/
        private void pictureBoxPatternSample_Paint(object sender, PaintEventArgs e)
        {
            e.Graphics.Clear(pictureBoxPatternSample.BackColor = Color.White);
            e.Graphics.SmoothingMode = SmoothingMode.AntiAlias;
            try
            {
                float a = (float)Properties.Settings.Default.PatternStep / 1000 * scale;
                float startAngle = (float)Properties.Settings.Default.PatternStartAngle / 57.3F;

                // Center point.
                PointF center = new PointF(pictureBoxPatternSample.ClientSize.Width / 2, pictureBoxPatternSample.ClientSize.Height / 2);
                PointF centerInMm = new PointF(pictureBoxPatternSample.ClientSize.Width / 2 * k / 1000, pictureBoxPatternSample.ClientSize.Height / 2 * k / 1000);

                // Draw axes.
                e.Graphics.DrawLine(Pens.Black, center.X, 0, center.X, pictureBoxPatternSample.ClientSize.Height);
                e.Graphics.DrawLine(Pens.Black, 0, center.Y, pictureBoxPatternSample.ClientSize.Width, center.Y);

                float maxRadius = (float)Properties.Settings.Default.PatternDiameter / 1000 / 2 * scale;

                List<PointF> points = new List<PointF>();

                switch (Properties.Settings.Default.PatternType)
                {
                    case (int)PatternType.Spiral:
                        points = spiralBuilder.GetSpiralPoints(centerInMm, a, startAngle, maxRadius).Select(p => new PointF(pictureBoxPatternSample.ClientSize.Width - p.X * 1000 / 3, p.Y * 1000 / 3)).ToList();
                        break;

                    case (int)PatternType.ZigZag:
                        points = zigZagBuilder.GetZigZagPoints(centerInMm, a, maxRadius * 2).Select(p => new PointF(p.X * 1000 / 3, p.Y * 1000 / 3)).ToList();
                        break;
                }

                e.Graphics.DrawLines(Pens.Red, points.ToArray());
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private void ButtonRunPattern_Click(object sender, EventArgs e)
        {
            Task.Factory.StartNew(() => patternDriver.RunConfiguredPattern());
            SetControlModeButtonsState(false);
        }

        private void SetControlModeButtonsState(bool enabled)
        {
            Invoke((MethodInvoker)delegate
            {
```

```
        SetPatternModeButtonsState(enabled);
        SetTransportationModeButtonsState(enabled);
    });
}

private void SetTransportationModeButtonsState(bool enabled)
{
    Invoke((MethodInvoker)delegate
    {
        buttonTransportationMode.Enabled = enabled;
        buttonHomeMode.Enabled = enabled;
    });
}

private void SetPatternModeButtonsState(bool enabled)
{
    if (this.Handle != null && patternDriver != null)
    {
        Invoke((MethodInvoker)delegate
        {
            buttonRunPattern.Enabled = enabled;
        });
    }
}

private void NumericUpDownTravelSpeed_ValueChanged(object sender, EventArgs e)
{
    Properties.Settings.Default.TravelSpeed = numericUpDownTravelSpeed.Value;
    Properties.Settings.Default.Save();
}

private void ButtonTransportationMode_Click(object sender, EventArgs e)
{
    LogMessage("Transportation mode initiated");
    Task.Factory.StartNew(() => patternDriver.RunTransportationModePattern());
    SetControlModeButtonsState(false);
}

private void TextBoxIpAddress_Leave(object sender, EventArgs e)
{
    Properties.Settings.Default.IpAddress = textBoxIpAddress.Text;
    Properties.Settings.Default.Save();
}

private void ButtonHomeMode_Click(object sender, EventArgs e)
{
    LogMessage("Home mode initiated");
    Task.Factory.StartNew(() => patternDriver.RunHomeModePattern());
    SetControlModeButtonsState(false);
}

private void ButtonStop_Click(object sender, EventArgs e)
{
    LogMessage("Stop All initiated");
    Task.Factory.StartNew(() => patternDriver.StopAll());
    buttonRunPattern.Enabled = true;
    buttonTransportationMode.Enabled = true;
    buttonHomeMode.Enabled = true;
}

private void buttonReset_Click(object sender, EventArgs e)
{
    Properties.Settings.Default.Reset();
    ReadSettings();
}

private void numericUpDownWeedColumns_Validating(object sender, System.ComponentModel.CancelEventArgs e)
{
    if (numericUpDownWeedColumns.Value * numericUpDownWeedInterval.Value > Properties.Settings.Default.BoxLength)
    {
```

```
                MessageBox.Show($"Multiplication of Weed columns and Interval must not exceed weed box length: {Properties.Settings.Default.BoxLength}", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                e.Cancel = true;
            }
            else
            {
                Properties.Settings.Default.WeedColumns = numericUpDownWeedColumns.Value;
                Properties.Settings.Default.Save();
            }
        }

        private void numericUpDownWeedInterval_Validating(object sender, System.ComponentModel.CancelEventArgs e)
        {
            if (numericUpDownWeedColumns.Value * numericUpDownWeedInterval.Value > Properties.Settings.Default.BoxLength)
            {
                MessageBox.Show($"Multiplication of Weed columns and Interval must not exceed weed box length: {Properties.Settings.Default.BoxLength}", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                e.Cancel = true;
            }
            else
            {
                Properties.Settings.Default.WeedInterval = numericUpDownWeedInterval.Value;
                Properties.Settings.Default.Save();
            }
        }

        private void checkBoxTakePhotos_CheckedChanged(object sender, EventArgs e)
        {
            Properties.Settings.Default.TakePhotos = checkBoxTakePhotos.Checked;
            Properties.Settings.Default.Save();

            SetPatternModeButtonsState(checkBoxTakePhotos.Checked);
        }

        private void comboBoxPatternType_SelectedIndexChanged(object sender, EventArgs e)
        {
            Properties.Settings.Default.PatternType = comboBoxPatternType.SelectedIndex;
            Properties.Settings.Default.Save();

            if(comboBoxPatternType.SelectedIndex == (int)PatternType.Spiral)
            {
                numericUpDownPatternStartAngle.Enabled = true;
                numericUpDownDeltaTheta.Enabled = true;
            }
            else
            {
                numericUpDownPatternStartAngle.Enabled = false;
                numericUpDownDeltaTheta.Enabled = false;
            }

            pictureBoxPatternSample.Refresh();
        }

        private void buttonSetPhotosFolderPath_Click(object sender, EventArgs e)
        {
            if (DialogResult.OK == folderBrowserDialog.ShowDialog())
            {
                Properties.Settings.Default.PhotosFolderPath = folderBrowserDialog.SelectedPath;
                Properties.Settings.Default.Save();
            }
        }
    }
}
```