9. pielikums

Eksperimentu stenda šablonu pārvaldnieka kods

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Threading;
using UR10.Patterns;

namespace UR10
{
   public interface IPatternDriverOut
   {
      void SendLogMessage(string message);
      void PatternCompleted();
   }

   public interface IPatternDriver
   {
      void RunConfiguredPattern();
      void EmergensyStopStatusReceived(bool status);
      void TurnOffAll();
      List<List<PointF>> GetExecutedPatterns();
      void RunTransportationModePattern();
      void RunHomeModePattern();
      void StopAll();
      void UpdateCartesianData(float x, float y, float z, float rX, float rY, float rZ);
      void CommandFinished();
      void SetPatternSpeed(float speed);
   }

   public class PatternDriver : IPatternDriver
   {
      private const float OutputVoltage = 10;
      private List<List<PointF>> executedPatterns = new List<List<PointF>>();

      private float actualX;
      private float actualY;
      private float actualZ;

      private float actualRX;
      private float actualRY;
      private float actualRZ;

      private const float HomeX = 0.16F;
      private const float HomeY = -0.22F;
      private const float HomeZ = 0.4F;

      private const float TransportationX = -0.43F;
      private const float TransportationY = -0.18F;
      private const float TransportationZ = 0.4F;

      //darba virsmas koordinates pret bazi
      //xy xy
      //01 11
      //00 10

      private const float ZMin = 0.4F;
      private const float ZMax = 0.6F;

      private const float X00 = 0.26F;
      private const float Y00 = -0.18F;

      private const float X10 = -0.6F;
```

```
private const float Y01 = -0.9F;

private const float xOffset = 0.04F;
private bool emergencyStoped = false;
private bool stopAll = false;
private IPatternDriverOut patternDriverUser;
private ISpiralBuilder spiralBuilder = new SpiralBuilder();
private IZigZagBuilder zigZagBuilder = new ZigZagBuilder();
private IPhotoCapturer photoCapturer = new PhotoCapturer();
private IAnnotationParser annotationParser = new AnnotationParser();
private ITcpServer tcpServer;
private bool commandFinished = false;
private bool executingPattern = false;

private List<string> sharedFileName = null;

public PatternDriver(ITcpServer tcpServer, IPatternDriverOut patternDriverUser)
{
   this.tcpServer = tcpServer;
   this.patternDriverUser = patternDriverUser;
}

public void CommandFinished()
{
   commandFinished = true;
}

public void EmergensyStopStatusReceived(bool status)
{
   emergencyStoped = status;
}

public void UpdateCartesianData(float x, float y, float z, float rX, float rY, float rZ)
{
   actualX = x;
   actualY = y;
   actualZ = z;

   actualRX = ConvertToDegrees(rX);
   actualRY = ConvertToDegrees(rY);
   actualRZ = ConvertToDegrees(rZ);

   if(executingPattern)
   {
      executedPatterns[executedPatterns.Count - 1].Add(new PointF(actualX, actualY));
   }
}

public void StopAll()
{
   stopAll = true;
   tcpServer.SendStopL();
   TurnOffAll();
}

public void RunConfiguredPattern()
{
   if (!Properties.Settings.Default.Row1Enabled &&
         !Properties.Settings.Default.Row2Enabled &&
         !Properties.Settings.Default.Row3Enabled &&
         !Properties.Settings.Default.Row4Enabled &&
         !Properties.Settings.Default.TakePhotos)
   {
      return;
   }

   executedPatterns.Clear();

   try
```

```
                {
                    CapturePhotos();

                    if(Properties.Settings.Default.TakePhotos)
                    {

                        List<BoundingBox> detectedWeedBoundingBoxes = annotationParser.ParseAnnotations(sharedFileName);
                        patternDriverUser.SendLogMessage($"Annotation parser retrieved {detectedWeedBoundingBoxes.Count} weed bounding
boxes");

                        tcpServer.SendDigitalOutOn();
                        WaitTillCommandFinished();

                        for (int box = 0; box < detectedWeedBoundingBoxes.Count; box++)
                        {
                            patternDriverUser.SendLogMessage($"Processing box No {box}...");

                            switch (Properties.Settings.Default.PatternType)
                            {
                                case (int)PatternType.Spiral:
                                    patternDriverUser.SendLogMessage($"Single spiral pattern is not implemented for box processing.");
                                    break;

                                case (int)PatternType.ZigZag:
                                    RunZigZagPatternSingle(detectedWeedBoundingBoxes[box]);
                                    break;
                            }
                        }
                    }
                    else
                    {
                        tcpServer.SendDigitalOutOn();
                        WaitTillCommandFinished();

                        if (Properties.Settings.Default.Row1Enabled)
                        {
                            switch (Properties.Settings.Default.PatternType)
                            {
                                case (int)PatternType.Spiral:
                                    RunSpiralPattern((float)Properties.Settings.Default.Row1Distance,
(float)Properties.Settings.Default.WeedInterval / 1000);
                                    break;

                                case (int)PatternType.ZigZag:
                                    RunZigZagPattern((float)Properties.Settings.Default.Row1Distance,
(float)Properties.Settings.Default.WeedInterval / 1000);
                                    break;
                            }
                        }

                        if (Properties.Settings.Default.Row2Enabled)
                        {
                            switch (Properties.Settings.Default.PatternType)
                            {
                                case (int)PatternType.Spiral:
                                    RunSpiralPattern((float)Properties.Settings.Default.Row2Distance,
(float)Properties.Settings.Default.WeedInterval / 1000);
                                    break;

                                case (int)PatternType.ZigZag:
                                    RunZigZagPattern((float)Properties.Settings.Default.Row2Distance,
(float)Properties.Settings.Default.WeedInterval / 1000);
                                    break;
                            }
                        }

                        if (Properties.Settings.Default.Row3Enabled)
                        {
                            switch (Properties.Settings.Default.PatternType)
                            {
```

```
                                case (int)PatternType.Spiral:
                                    RunSpiralPattern((float)Properties.Settings.Default.Row3Distance,
(float)Properties.Settings.Default.WeedInterval / 1000);
                                    break;

                                case (int)PatternType.ZigZag:
                                    RunZigZagPattern((float)Properties.Settings.Default.Row3Distance,
(float)Properties.Settings.Default.WeedInterval / 1000);
                                    break;
                            }
                        }

                        if (Properties.Settings.Default.Row4Enabled)
                        {
                            switch (Properties.Settings.Default.PatternType)
                            {
                                case (int)PatternType.Spiral:
                                    RunSpiralPattern((float)Properties.Settings.Default.Row4Distance,
(float)Properties.Settings.Default.WeedInterval / 1000);
                                    break;

                                case (int)PatternType.ZigZag:
                                    RunZigZagPattern((float)Properties.Settings.Default.Row4Distance,
(float)Properties.Settings.Default.WeedInterval / 1000);
                                    break;
                            }
                        }
                    }

                    tcpServer.SendDigitalOutOff();
                    WaitTillCommandFinished();

                    //CapturePhotos();

                    RunHomeModePattern();
                }
                catch (Exception e)
                {
                    patternDriverUser.SendLogMessage("System exception. Executing emergency Laser turn off and Go home commands");
                    TurnOffAll();
                    RunHomeModePattern();
                }
            }
            public void TurnOffAll()
            {
                tcpServer.SendDigitalOutOff();
                WaitTillCommandFinished();
                tcpServer.SendAnalogOutOff();
                WaitTillCommandFinished();
            }

            public List<List<PointF>> GetExecutedPatterns()
            {
                return executedPatterns;
            }

            public void RunTransportationModePattern()
            {
                string error = SendMoveLCommand(TransportationX, TransportationY, TransportationZ,
Properties.Settings.Default.TravelSpeed / 1000, 180, 0, 0);

                if (string.IsNullOrEmpty(error))
                {
                    WaitTillCommandFinished();

                    patternDriverUser.SendLogMessage("Transportation mode enabled");
                    patternDriverUser.PatternCompleted();
                }
                else
                {
```

```
        patternDriverUser.SendLogMessage(error);
    }
}

public void RunHomeModePattern()
{
    string error = SendMoveLCommand(HomeX, HomeY, HomeZ, Properties.Settings.Default.TravelSpeed / 1000, 180, 0, 0);

    if (string.IsNullOrEmpty(error))
    {
        WaitTillCommandFinished();

        patternDriverUser.SendLogMessage("Home mode enabled");
        patternDriverUser.PatternCompleted();
    }
    else
    {
        patternDriverUser.SendLogMessage(error);
    }
}

public void SetPatternSpeed(float speed)
{
    string error = SendSetPatternSpeedCommand(speed);
    if (string.IsNullOrEmpty(error))
    {
        WaitTillCommandFinished();

        patternDriverUser.SendLogMessage("Pattern speed is set");
    }
    else
    {
        patternDriverUser.SendLogMessage(error);
    }
}

public enum PatternType
{
    Spiral,
    ZigZag
}

private List<PointF> CalculateSpiralPattern(PointF center)
{
    float a = (float)Properties.Settings.Default.PatternStep / 1000;
    float startAngle = (float)Properties.Settings.Default.PatternStartAngle / 57.3F;
    float maxRadius = (float)Properties.Settings.Default.PatternDiameter / 1000 / 2;

    return spiralBuilder.GetSpiralPoints(center, a, startAngle, maxRadius);
}

private List<PointF> CalculateZigZagPattern(PointF center, float maxWidth = 0, float maxHeight = 0)
{
    float step = (float)Properties.Settings.Default.PatternStep / 1000;
    maxWidth = maxWidth == 0 ? (float)Properties.Settings.Default.PatternDiameter / 1000 : maxWidth / 1000;

    return zigZagBuilder.GetZigZagPoints(center, step, maxWidth, maxHeight / 1000);
}

private bool RunSpiralPattern(float y, float xStep)
{
    for (int column = 0; column < Properties.Settings.Default.WeedColumns; column++)
    {
        float xCenter = X00 - column * xStep - xOffset;
        float yCenter = y / 1000;

        bool isSpiralCenterReached = TravelToCoordinate(new PointF(xCenter, yCenter));

        if (isSpiralCenterReached)
        {
```

```
            List<PointF> points = CalculateSpiralPattern(new PointF(xCenter, yCenter));
            executedPatterns.Add(new List<PointF>());
            bool patternCompleted = RunPatterByPoints(points, CommandType.MoveP);

            if (!patternCompleted)
            {
                return false;
            }
        }
        else
        {
            return false;
        }
    }

    return true;
}

private bool RunZigZagPattern(float y, float xStep)
{
    for (int column = 0; column < Properties.Settings.Default.WeedColumns; column++)
    {
        float xCorner = X00 - column * xStep - xOffset + (float)Properties.Settings.Default.PatternDiameter / 2 / 1000;
        float yCorner = y / 1000 + (float)Properties.Settings.Default.PatternDiameter / 2 / 1000;

        bool isZigZagCornerReached = TravelToCoordinate(new PointF(xCorner, yCorner));

        if (isZigZagCornerReached)
        {
            List<PointF> points = CalculateZigZagPattern(
                new PointF(
                    xCorner - (float)Properties.Settings.Default.PatternDiameter / 2 / 1000,
                    yCorner - (float)Properties.Settings.Default.PatternDiameter / 2/ 1000)
                );
            executedPatterns.Add(new List<PointF>());
            bool patternCompleted = RunPatterByPoints(points, CommandType.MoveL);

            if (!patternCompleted)
            {
                return false;
            }
        }
        else
        {
            return false;
        }
    }

    return true;
}

private bool RunZigZagPatternSingle(BoundingBox box)
{
    float xCorner = ((float)Properties.Settings.Default.BaseXCoordinate + box.Xmin) / 1000;
    float yCorner = ((float)Properties.Settings.Default.BaseYCoordinate + box.Ymin) / 1000;

    bool isZigZagCornerReached = TravelToCoordinate(new PointF(xCorner, yCorner));

    if (isZigZagCornerReached)
    {
        List<PointF> points = CalculateZigZagPattern(
            new PointF(
                xCorner - (box.Xmin - box.Xmax) / 2 / 1000,
                yCorner + (box.Ymax - box.Ymin)/2/1000),
            box.Xmin-box.Xmax,
            box.Ymax-box.Ymin
            );
        executedPatterns.Add(new List<PointF>());
        bool patternCompleted = RunPatterByPoints(points, CommandType.MoveL);
```

```
            if (!patternCompleted)
            {
                return false;
            }
        }
        else
        {
            return false;
        }

        return true;
    }

    private bool CapturePhotos()
    {
        if (Properties.Settings.Default.TakePhotos)
        {
            sharedFileName = new List<string>();

            float x = -0.58F;
            float y = -0.72F;
            float z = 0.45F;

            float xStep = 0.26F;
            float yStep = 0.25F;

            float rX = 127;
            float rY = 0;
            float rZ = 127;

            int maxRowIndex = Properties.Settings.Default.Row3Enabled || Properties.Settings.Default.Row4Enabled ? 2 : 1;
            int currentRowIndex = Properties.Settings.Default.Row1Enabled || Properties.Settings.Default.Row2Enabled ? 0 : 1;

            for (int row = currentRowIndex; row < maxRowIndex; row++)
            {
                for (int col = 0; col < 3; col++)
                {
                    float xx = x + col * xStep;
                    float yy = y + row * yStep;

                    string error = SendMoveLCommand(xx, yy, z, Properties.Settings.Default.TravelSpeed / 1000, rX, rY, rZ);

                    if (string.IsNullOrEmpty(error))
                    {
                        WaitTillCommandFinished();

                        patternDriverUser.SendLogMessage($"Photo position x={x} y={y} z={z} reached. Capturing photo");
                        (string output, string errors) = photoCapturer.CapturePhoto();

                        if (string.IsNullOrEmpty(errors))
                        {
                            patternDriverUser.SendLogMessage(output);

                            //string moved = photoCapturer.OrganizePhotoFiles(DateTime.Now.ToShortDateString().Replace("/", "-"));
                            string movedFileName = photoCapturer.OrganizePhotoFiles(Properties.Settings.Default.PhotosFolderPath, col +
row * 3);

                            if (movedFileName.StartsWith("Error:"))
                            {
                                patternDriverUser.SendLogMessage(movedFileName);
                                return false;
                            }
                            else
                            {
                                sharedFileName.Add(movedFileName);
                            }
                        }
                        else
                        {
                            patternDriverUser.SendLogMessage(errors);
```

```
                    return false;
                }
            }
            else
            {
                patternDriverUser.SendLogMessage(error);
                return false;
            }
        }
    }
}

return true;
}

private bool TravelToCoordinate(PointF center)
{
patternDriverUser.SendLogMessage("Traveling to coordinate");

string error = SendMoveLCommand(center.X, center.Y, HomeZ, Properties.Settings.Default.TravelSpeed / 1000, 180, 0, 0);

if (string.IsNullOrEmpty(error))
{
    WaitTillCommandFinished();

    patternDriverUser.SendLogMessage("Coordinate is reached");
    return true;
}
else
{
    patternDriverUser.SendLogMessage(error);
    return false;
}
}

private bool RunPatterByPoints(List<PointF> points, CommandType type)
{
string error = null;

patternDriverUser.SendLogMessage("Pattern started");

for (int x = 0; x < points.Count; x++)
{
    error = StorePositionPoint(points[x].X, points[x].Y, ZMin, 180, 0, 0);

    if (string.IsNullOrEmpty(error))
    {
        WaitTillCommandFinished();
    }
    else
    {
        patternDriverUser.SendLogMessage(error);
        return false;
    }
}

tcpServer.SendAnalogOutOn(OutputVoltage);
WaitTillCommandFinished();

executingPattern = true;

if (type == CommandType.MoveP)
{
    tcpServer.ExecuteStoredMoveP();
}
else if (type == CommandType.MoveL)
{
    tcpServer.ExecuteStoredMoveL();
}
```

```
            WaitTillCommandFinished();
            executingPattern = false;

            tcpServer.SendAnalogOutOff();
            WaitTillCommandFinished();

            if (string.IsNullOrEmpty(error))
            {
                patternDriverUser.SendLogMessage("Pattern completed");
            }
            return true;
        }

        private string SendMoveLCommand(float x, float y, float z, decimal speed, float rX, float rY, float rZ)
        {
            if (stopAll)
            {
                return "Stop";
            }

            if (x <= X00 && x >= X10 && y <= Y00 && y >= Y01 && z >= ZMin && z <= ZMax && speed <= 0.2M)
            {
                tcpServer.SendMoveL(new float[] { x, y, z, ConvertToRadians(rX), ConvertToRadians(rY), ConvertToRadians(rZ) },
(float)speed);
                return null;
            }
            else
            {
                return $"Bad movel command. x={x}, y={y}, z={z}, speed={speed}";
            }
        }

        private string SendSetPatternSpeedCommand(float speed)
        {
            if (stopAll)
            {
                return "Stop";
            }

            if (speed <= 0.2F)
            {
                tcpServer.SetPatternSpeed(speed);
                return null;
            }
            else
            {
                return $"Bad set pattern speed command. Speed={speed}";
            }
        }

        private string SendMovePCommand(float x, float y, float z, decimal speed, float rX, float rY, float rZ)
        {
            if (stopAll)
            {
                return "Stop";
            }

            if (x <= X00 && x >= X10 && y <= Y00 && y >= Y01 && z >= ZMin && z <= ZMax && speed <= 0.2M)
            {
                tcpServer.SendMoveP(new float[] { x, y, z, ConvertToRadians(rX), ConvertToRadians(rY), ConvertToRadians(rZ) },
(float)speed);
                return null;
            }
            else
            {
                return $"Bad movep command. x={x}, y={y}, z={z}, speed={speed}";
            }
        }

        private string StorePositionPoint(float x, float y, float z, float rX, float rY, float rZ)
```

```
        {
            if (stopAll)
            {
                return "Stop";
            }

            if (x <= X00 && x >= X10 && y <= Y00 && y >= Y01 && z >= ZMin && z <= ZMax)
            {
                tcpServer.StorePositionPoint(new float[] { x, y, z, ConvertToRadians(rX), ConvertToRadians(rY), ConvertToRadians(rZ) });
                return null;
            }
            else
            {
                return $"Bad store movep command. x={x}, y={y}, z={z}";
            }
        }

        private float ConvertToRadians(double angle)
        {
            return (float)(Math.PI / 180 * angle);
        }

        private float ConvertToDegrees(double angle)
        {
            return (float)(angle * 180 / Math.PI);
        }

        private void WaitTillCommandFinished()
        {
            while (!commandFinished)
            {
                Thread.Sleep(1);
            }
            commandFinished = false;
        }

        private enum CommandType
        {
            MoveL,
            MoveP
        }
    }
}
```