



Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Hálózati Rendszerek és Szolgáltatások Tanszék

Dénes Csaba Antal

SZIMULÁCIÓS KÖNYVTÁR KVANTUMMŰHOLDAS RENDSZEREK TERVEZÉSÉRE ÉS IRÁNYÍTÁSÁRA

– FÜGGELÉKEK –

TANSZÉKI KONZULENSEK

Dr. Bacsárdi László

Mihály András

KÜLSŐ KONZULENSEK

Dr. Takarics Béla

Dr. Vanek Bálint

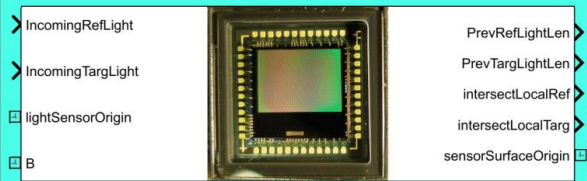
BUDAPEST, 2024

Tartalomjegyzék

Optikai részegységek modul szintű ábrázolása	3
Optikai alegység belső felépítése	4
Optikai lencsék szabad alakítása	5
Orientációs alegység moduljai	9
Orientációs és pozicionáló egység felépítése	10
Nyomaték vezérelt reakció kerék rendszer.....	10

Optikai részegységek modul szintű ábrázolása

Optical Head Structures



Light Sensor Raw



OneBoard Laser Source



Filter Lens Unidirectional



Optical Lens



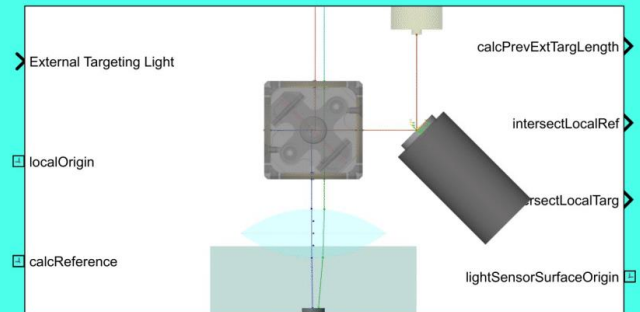
Filter Lens Bidirectional



Mirror Lens



FSM mirror

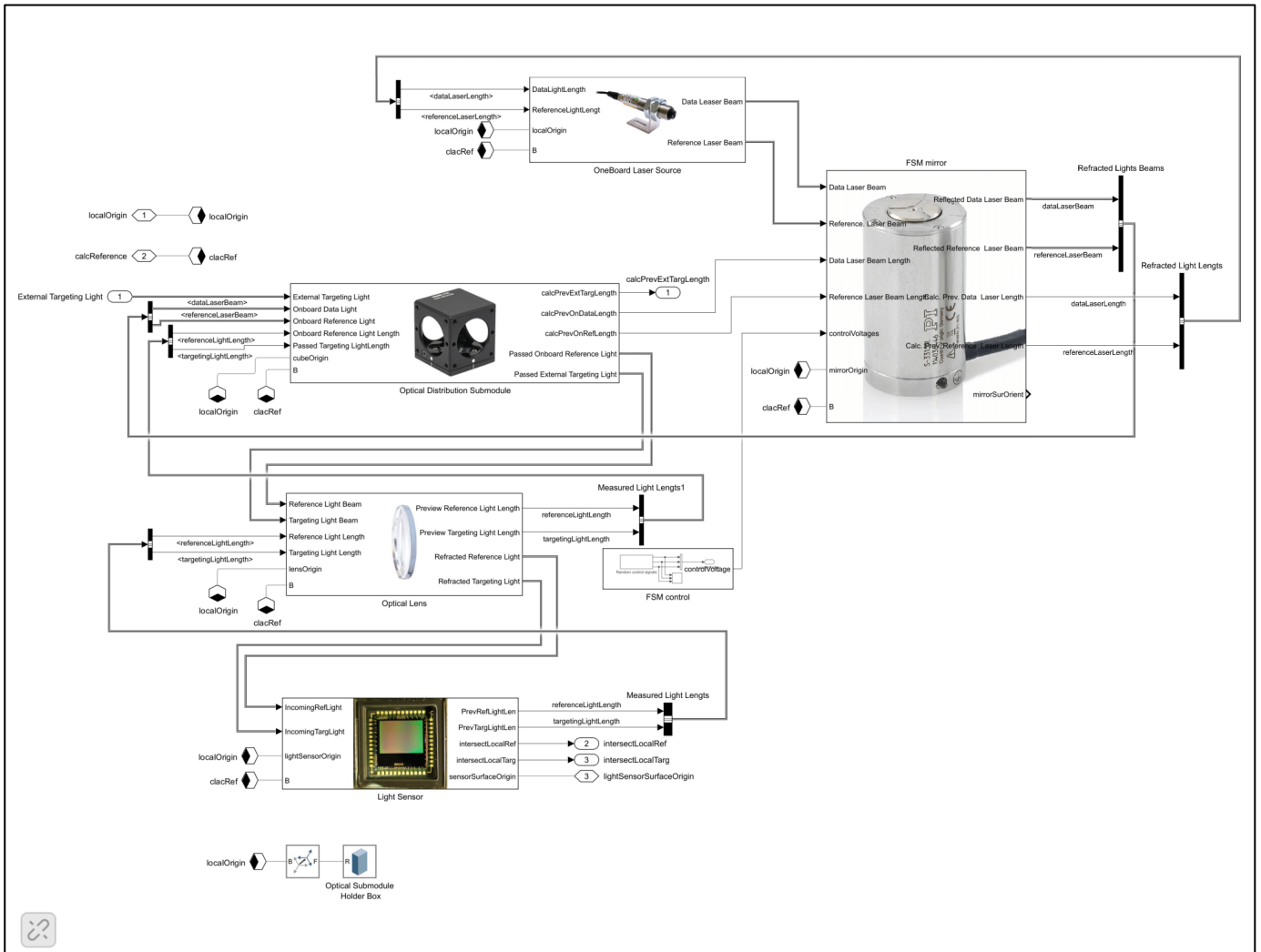


Optical Targeting Submodule



Dicroic Mirror with Filter Lens

Optikai alegység belső felépítése



Optikai lencsék szabad alakítása

```
classdef focusingLensInitializationScript

    methods(Static)

        % Following properties of 'maskInitContext' are available to use:
        % - BlockHandle
        % - MaskObject
        % - MaskWorkspace: Use get/set APIs to work with mask workspace.
        function MaskInitialization(maskInitContext)
            %% Get the variables on mask
            lensType = get_param(gcb,"lensType");
            lensRadiusType = get_param(gcb,"lensCurvSelector");
            focallength = str2double(get_param(gcb,"focallength"));
            refractiveIndexLens =
str2double(get_param(gcb,"refractiveIndexLensMaterial"));
            refractiveIndexEnvironment =
str2double(get_param(gcb,"refractiveIndexEnvironment"));
            LensWidthFront = str2double(get_param(gcb,"LensWidthFront"));
            LensWidthBack = str2double(get_param(gcb,"LensWidthBack"));
            generatorAccurancy = str2double(get_param(gcb,"LensWidthAcurrancy"));
            lensBaseWidth = str2double(get_param(gcb,"lensBaseWidth"));
            curvatureRadiusFront = 0;
            curvatureRadiusBack = 0;
            maskObject = Simulink.Mask.get(gcb);
            ws = maskInitContext.MaskWorkspace;

            %% Calculation of lens Radius
            switch lensRadiusType
                case "Calculated"
                    if(strcmp(lensType,"different curvature lens"))
                        curvatureRadiusFront = abs((refractiveIndexLens-1)*focallength);
                        curvatureRadiusBack = abs((refractiveIndexLens-1)*focallength);
                        %% Write out the lens radius
                        maskObject.getParameter('calcCurvatureRadiusFront').Value =
num2str(curvatureRadiusFront);
                        maskObject.getParameter('calcCurvatureRadiusBack').Value =
num2str(curvatureRadiusBack);
                    else
                        %% Symetric Lens
                        % Because is in symetric case, get the symetric lens slice
width
                        tempLensWidthFront = get_param(gcb,'value@LensWidthSym');
                        tempLensWidthBack = get_param(gcb,'value@LensWidthSym');
                        % Get the lens form
                        tempSurfaceTypeFront =
get_param(gcb,'value@SurfaceTypeSym'); % 1 - convex, 2- concave, 3 - plane
                        % Get the width of lens
                        fullLensWidth = tempLensWidthFront + tempLensWidthBack +
lensBaseWidth;
                        % Get the sign of the radius calculation (positiv for convex
lenses, negativ for concave lenses)
                        switch tempSurfaceTypeFront
                            case 1
                                radiusCalcSign = 1;
                            case 2
                                radiusCalcSign = -1;
                            otherwise
                                radiusCalcSign = 0;
                        end

                        % Create the quadratic components
                        a = 1/focallength;
                    end
                end
            end
        end
    end
end
```

```

        b = -2*(refractiveIndexLens - refractiveIndexEnvironment)
        c = ((refractiveIndexLens -
refractiveIndexEnvironment)^2*fullLensWidth)/refractiveIndexLens;
        % Calculate the value of R
        deltaRoot = sqrt(b^2-4*a*c);
        if(deltaRoot>=0)
            curvatureRadiusFront = (-
b+radiusCalcSign*deltaRoot)/(2*a);
            curvatureRadiusBack = curvatureRadiusFront;
        else
            % Todo: Handle error case
        end
        maskObject.getParameter('calcCurvatureRadiusSym').Value =
num2str(curvatureRadiusFront);
    end

    case "Constant Values"
        if(strcmp(lensType,"different curvature lens"))
            %% Read writed value
            curvatureRadiusFront =
str2num(get_param(gcb,"calcCurvatureRadiusFront"));
            curvatureRadiusBack =
str2num(get_param(gcb,"calcCurvatureRadiusBack"));
        else
            curvatureRadiusFront =
str2num(get_param(gcb,"calcCurvatureRadiusSym"));
            curvatureRadiusBack =
str2num(get_param(gcb,"calcCurvatureRadiusSym"));
        end
    end

    %% Save the lens 3D point cloud to masc workspace for calculation
    templensType = get_param(gcb,'value@lensType');
    switch templensType
        case 1 %different
            tempSurfaceTypeFront = get_param(gcb,'value@SurfaceTypeFront');
            tempSurfaceTypeBack = get_param(gcb,'value@SurfaceTypeBack');
            tempLensWidthFront = get_param(gcb,'value@LensWidthFront');
            tempLensWidthBack = get_param(gcb,'value@LensWidthBack');
            tempcurvatureRadiusFront =
get_param(gcb,'value@calcCurvatureRadiusFront');
            tempcurvatureRadiusBack =
get_param(gcb,'value@calcCurvatureRadiusBack');
        case 2 %symmetric
            tempSurfaceTypeFront = get_param(gcb,'value@SurfaceTypeSym');
            tempSurfaceTypeBack = get_param(gcb,'value@SurfaceTypeSym');
            tempLensWidthFront = get_param(gcb,'value@LensWidthSym');
            tempLensWidthBack = get_param(gcb,'value@LensWidthSym');
            tempcurvatureRadiusFront =
get_param(gcb,'value@calcCurvatureRadiusSym');
            tempcurvatureRadiusBack =
get_param(gcb,'value@calcCurvatureRadiusSym');
        end
        ws.set('tempSurfaceTypeFront',tempSurfaceTypeFront);
        ws.set('tempSurfaceTypeBack',tempSurfaceTypeBack);
        ws.set('tempLensWidthFront',tempLensWidthFront);
        ws.set('tempLensWidthBack',tempLensWidthBack);
        ws.set('tempcurvatureRadiusFront',tempcurvatureRadiusFront);
        ws.set('tempcurvatureRadiusBack',tempcurvatureRadiusBack);

    %% Calculate lens base radius

```

```

        halfFrontLensThickness = curvatureRadiusFront*sin(acos((curvatureRadiusFront -
templensWidthFront)/curvatureRadiusFront));
        halfBackLensThickness = curvatureRadiusBack*sin(acos((curvatureRadiusBack -
templensWidthBack)/curvatureRadiusBack));

        %% Save half thicknesses to mask workspace in case if on slice is plane (not
convex or concave)
        ws.set('halfFrontLensThickness',halfFrontLensThickness);
        ws.set('halfBackLensThickness',halfBackLensThickness);

        %% Select the biggest Slice Radius
        lensBaseRadius = max(halfFrontLensThickness, halfBackLensThickness);
        %ws.set('lensBaseRadius',lensBaseRadius);
        switch tempSurfaceTypeFront
            case 1
                lensFrontObject2DPoints =
getLensSurfaceProjection(tempcurvatureRadiusFront,templensWidthFront,generatorAccuracy,fa
lse);
            case 2
                lensFrontObject2DPoints =
getLensSurfaceProjection(tempcurvatureRadiusFront,templensWidthFront,generatorAccuracy,tr
ue);
            case 3
                lensFrontObject2DPoints = [ 0 templensWidthFront;lensBaseRadius
templensWidthFront];
            end
            ws.set('frontLensObject3DPoints',lensFrontObject2DPoints);

            switch tempSurfaceTypeBack
                case 1
                    lensBackObject2DPoints =
getLensSurfaceProjection(tempcurvatureRadiusBack,templensWidthBack,generatorAccuracy,fals
e);
                case 2
                    lensBackObject2DPoints =
getLensSurfaceProjection(tempcurvatureRadiusBack,templensWidthBack,generatorAccuracy,true
);
                case 3
                    lensBackObject2DPoints = [ 0 templensWidthBack;lensBaseRadius
templensWidthBack];
            end

            ws.set('backLensObject3DPoints',lensBackObject2DPoints);
            %% Transfer points to workspace for checking
            assignin('base','lensFrontObject2DPoints',lensFrontObject2DPoints);
            assignin('base','lensBackObject2DPoints',lensBackObject2DPoints);
        end

function lensType(callbackContext)
    maskObj = Simulink.Mask.get(gcb);
    ParamObj = callbackContext.ParameterObject;
    myParam = get_param(gcb,'lensType');
    myEnum = {ParamObj.TypeOptions.EnumerationMembers.DescriptiveName};
    switch find(string(myEnum)==myParam)
        case 1 %different
            maskObj.getDialogControl('Container_symmetric').Visible = "off";
            maskObj.getDialogControl('Container_back').Visible = "on";
            maskObj.getDialogControl('Container_front').Visible = "on";
        case 2 %symmetric
            maskObj.getDialogControl('Container_symmetric').Visible = "on";
            maskObj.getDialogControl('Container_back').Visible = "off";
            maskObj.getDialogControl('Container_front').Visible = "off";
    end
end

```

```

end

function SurfaceTypeSym(callbackContext)
    maskObj = Simulink.Mask.get(gcf);
    ParamObj = callbackContext.ParameterObject;
    myParam = get_param(gcf, "SurfaceTypeSym");
    myEnum = {ParamObj.TypeOptions.EnumerationMembers.DescriptiveName};
    switch find(string(myEnum)==myParam)
        case {1,2} %convex,%concave
            maskObj.getParameter('calcCurvatureRadiusSym').Visible = "on";
        case 3 %plane
            maskObj.getParameter('calcCurvatureRadiusSym').Visible = "off";
    end
end

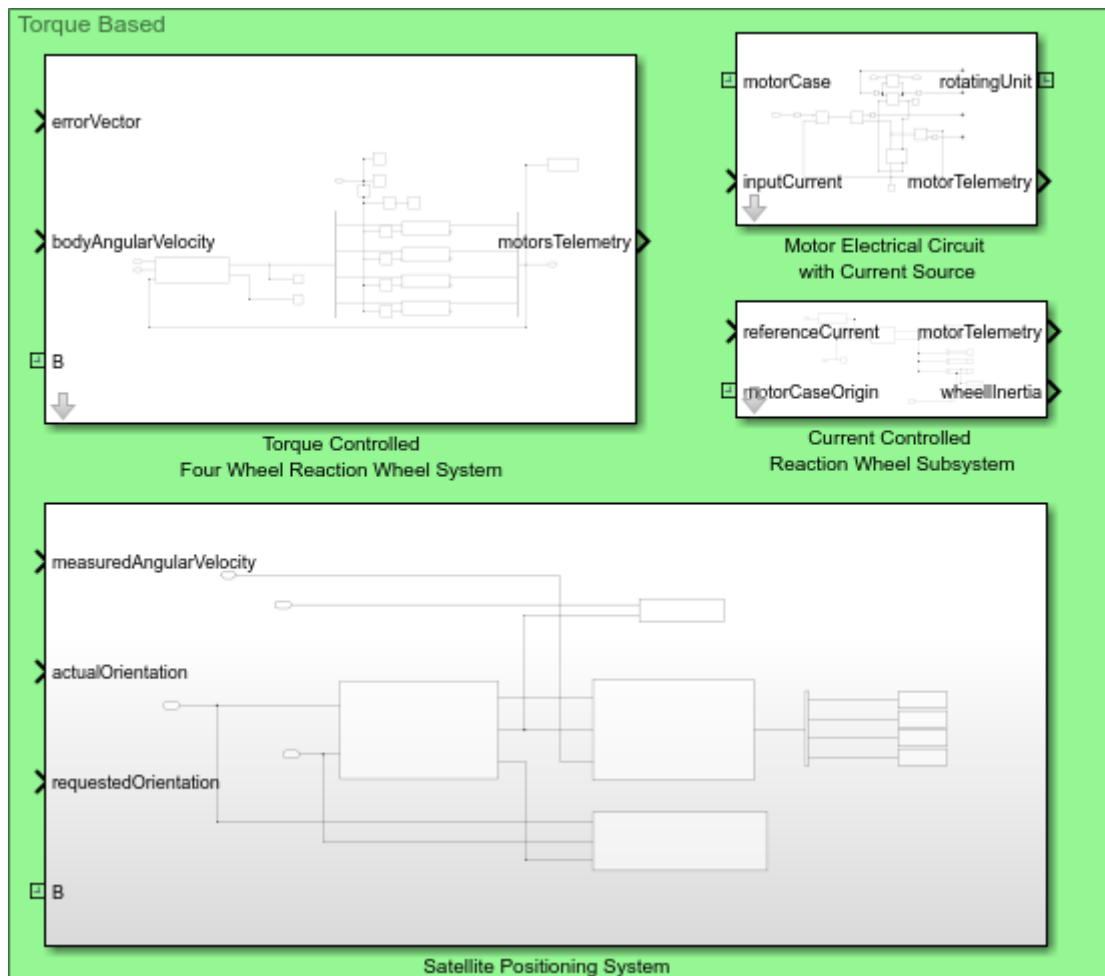
function SurfaceTypeFront(callbackContext)
    maskObj = Simulink.Mask.get(gcf);
    ParamObj = callbackContext.ParameterObject;
    myParam = get_param(gcf, "SurfaceTypeFront");
    myEnum = {ParamObj.TypeOptions.EnumerationMembers.DescriptiveName};
    switch find(string(myEnum)==myParam)
        case {1,2} %convex,%concave
            maskObj.getParameter('calcCurvatureRadiusFront').Visible = "on";
        case 3 %plane
            maskObj.getParameter('calcCurvatureRadiusFront').Visible = "off";
    end
end

function SurfaceTypeBack(callbackContext)
    maskObj = Simulink.Mask.get(gcf);
    ParamObj = callbackContext.ParameterObject;
    myParam = get_param(gcf, "SurfaceTypeBack");
    myEnum = {ParamObj.TypeOptions.EnumerationMembers.DescriptiveName};
    switch find(string(myEnum)==myParam)
        case {1,2} %convex,%concave
            maskObj.getParameter('calcCurvatureRadiusBack').Visible = "on";
        case 3 %plane
            maskObj.getParameter('calcCurvatureRadiusBack').Visible = "off";
    end
end

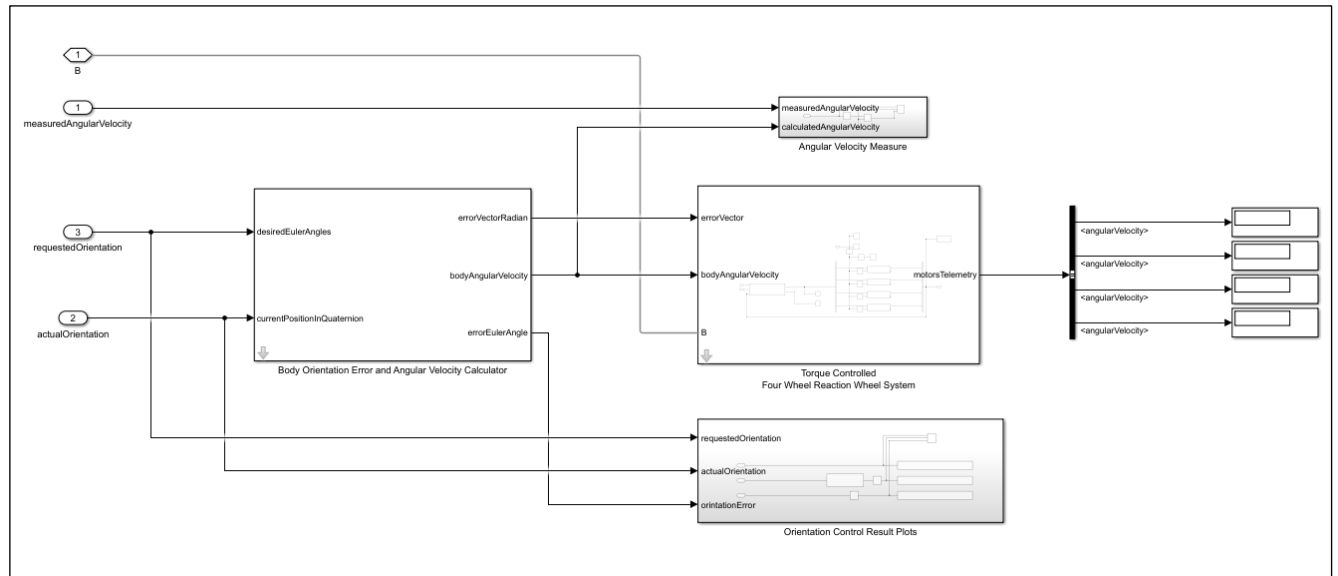
function lensCurvSelector(callbackContext)
    maskObj = Simulink.Mask.get(gcf);
    ParamObj = callbackContext.ParameterObject;
    myParam = get_param(gcf, "lensCurvSelector");
    myEnum = {ParamObj.TypeOptions.EnumerationMembers.DescriptiveName};
    switch find(string(myEnum)==myParam)
        case 1 % calc
            %% Set the Lens Radius Parameter Line to Disables, if lens curvature
            radius selector is Calculated
            maskObj.getParameter('calcCurvatureRadiusSym').Enabled = 'off';
            maskObj.getParameter('calcCurvatureRadiusFront').Enabled = 'off';
            maskObj.getParameter('calcCurvatureRadiusBack').Enabled = 'off';
        case 2 % const
            %% Set the Lens Radius Parameter Line to Enabled, to write in the
            value
            maskObj.getParameter('calcCurvatureRadiusSym').Enabled = 'on';
            maskObj.getParameter('calcCurvatureRadiusFront').Enabled = 'on';
            maskObj.getParameter('calcCurvatureRadiusBack').Enabled = 'on';
    end
end
end
end
end
end

```

Orientációs alegység moduljai



Orientációs és pozícionáló egység felépítése



Nyomaték vezérelt reakció kerék rendszer

