```cpp
#include "MotionAdaptationToolV2HandlerDialog.h"
#include "ui_MotionAdaptationToolV2HandlerDialog.h"


#include <MMM/Exceptions.h>
#include <MMM/Motion/MotionRecording.h>
#include <MMM/Model/ModelReaderXML.h>
#include <QFileDialog>
#include <QCheckBox>
#include <QMessageBox>
#include <VirtualRobot/Robot.h>
#include <Inventor/nodes/SoMatrixTransform.h>
#include <Inventor/nodes/SoUnits.h>
#include <MMM/Motion/Plugin/ModelPosePlugin/ModelPoseSensor.h>
#include <MMM/Motion/Plugin/ModelPosePlugin/ModelPoseSensorMeasurement.h>
#include <MMM/Motion/Plugin/KinematicPlugin/KinematicSensorMeasurement.h>
#include <MMM/Motion/Plugin/KinematicPlugin/KinematicSensor.h>
#include <SimoxUtility/math/convert.h>
#include <VirtualRobot/IK/DiffIK/CompositeDiffIK.h>
#include <VirtualRobot/Visualization/VisualizationNode.h>
#include <VirtualRobot/Visualization/TriMeshModel.h>
#include <VirtualRobot/Visualization/CoinVisualization/CoinVisualizationFactory.h>
#include <VirtualRobot/Visualization/TriMeshUtils.h>
#include <Inventor/nodes/SoCoordinate3.h>
#include <Inventor/nodes/SoDrawStyle.h>
#include <Inventor/nodes/SoPointSet.h>
#include <Inventor/SbVec3f.h>
#include <QPushButton>
#include <QFileDialog>
#include <set>


MotionAdaptationToolV2HandlerDialog::MotionAdaptationToolV2HandlerDialog(QWidget* parent,
MMM::MotionRecordingPtr motions) :
    QDialog(parent),
    ui(new Ui::MotionAdaptationToolV2HandlerDialog),
    motions(motions),
    currentObject(nullptr)
{
    ui->setupUi(this);


    // map gui elements to methods
    connect(ui->ChooseMotionComboBox, SIGNAL(currentTextChanged(const QString&)), this,
SLOT(setCurrentMotion(const QString&)));
    connect(ui->storeButton, &QPushButton::clicked, this,
&MotionAdaptationToolV2HandlerDialog::storeMotion);
    loadMotions();
}


MotionAdaptationToolV2HandlerDialog::~MotionAdaptationToolV2HandlerDialog() {
    delete ui;
}


void MotionAdaptationToolV2HandlerDialog::jumpTo(float timestep) {
    // TODO - Only an example: Adapt your new object poses and add the corresponding mmm motion from
you motion representation here
    for (auto object : objects) {
        // Set and visualize object motion
        auto motion = motions->getMotion(object.first);
        // Retrieve objct root pose from recording for given timestep
        auto modelPoseSensor = motion->getSensorByType<MMM::ModelPoseSensor>();
        auto modelPoseSensorMeasurement = modelPoseSensor->getDerivedMeasurement(timestep);
        if (modelPoseSensorMeasurement) {
            // normally interpolated, but if timestep is less than start or bigger than end the
measurement is a nullptr
            object.second->setGlobalPose(modelPoseSensorMeasurement->getRootPose());
        }
    }
    auto motion = motions->getReferenceModelMotion();
    if (motion) {
        auto modelPoseSensor = motion->getSensorByType<MMM::ModelPoseSensor>();
        auto kinematicSensor = motion->getSensorByType<MMM::KinematicSensor>();
        auto modelPoseSensorMeasurement = modelPoseSensor->getDerivedMeasurement(timestep);
        if (!modelPoseSensorMeasurement) return;
        auto rootPose = modelPoseSensorMeasurement->getRootPose();
        // set root pose from motion recording
        mmm_visualized->setGlobalPose(rootPose);
        kinematicSensor->initializeModel(mmm, timestep); // sets the joint angles
```

```
        // Only examples, run RobotViewer (just type in terminal) and select
repos/mmmtools/data/Model/Winter/mmm.xml to find out the positions and names
        auto leftHandPose_RootFrame = mmm->getRobotNode("Hand L TCP")->getPoseInRootFrame();
        auto leftEllbowPose_RootFrame = mmm->getRobotNode("LEsegment_joint")->getPoseInRootFrame();
        auto rightKneePose_RootFrame = mmm->getRobotNode("RKsegment_joint")->getPoseInRootFrame();
        auto leftKneePose_RootFrame = mmm->getRobotNode("LKsegment_joint")->getPoseInRootFrame();
        auto rightFootPose_RootFrame = mmm->getRobotNode("RightFootHeight_joint")-
>getPoseInRootFrame();
        auto leftFootPose_RootFrame = mmm->getRobotNode("LeftFootHeight_joint")-
>getPoseInRootFrame();


        // Initialize and solve IK to match this pose with the given robot node set. It can be
changed by you to include the relevant joints, take a look at mmm.xml
        auto robotNodeSet = mmm_visualized->getRobotNodeSet("BodyLeftArm");
        VirtualRobot::CompositeDiffIK cik(robotNodeSet);


        // Add targets fr inverse kinematic, these end effector poses have to be satisfied
        // VirtualRobot::IKSolver::CartesianSelection::All means Position and Orientation - Can be
changed if desird
        auto target_left_hand = cik.addTarget(mmm_visualized->getRobotNode("Hand L TCP"),
leftHandPose_RootFrame, VirtualRobot::IKSolver::CartesianSelection::All);
        auto target_left_foot = cik.addTarget(mmm_visualized->getRobotNode("LeftFootHeight_joint"),
leftFootPose_RootFrame, VirtualRobot::IKSolver::CartesianSelection::All);
        auto target_right_foot = cik.addTarget(mmm_visualized-
>getRobotNode("RightFootHeight_joint"), rightFootPose_RootFrame,
VirtualRobot::IKSolver::CartesianSelection::All);


        // Add nullspace target - adapt the nullspace to satisfy these targets as best as possible
        // Here, only the position is used and not the orientation - Can be changed if desired,
similar to above, look at Class NullspaceTarget
        // kp is a weight value for this nullspace adaption - higher value means that it will be
considered more
        VirtualRobot::CompositeDiffIK::NullspaceTargetPtr n1(new
VirtualRobot::CompositeDiffIK::NullspaceTarget(robotNodeSet, mmm_visualized-
>getRobotNode("LEsegment_joint"), simox::math::mat4f_to_pos(leftEllbowPose_RootFrame)));
        n1->kP = 0.3;
        cik.addNullspaceGradient(n1);
        VirtualRobot::CompositeDiffIK::NullspaceTargetPtr n2(new
VirtualRobot::CompositeDiffIK::NullspaceTarget(robotNodeSet, mmm_visualized-
>getRobotNode("LKsegment_joint"), simox::math::mat4f_to_pos(leftKneePose_RootFrame)));
        n2->kP = 0.3;
        cik.addNullspaceGradient(n2);
        VirtualRobot::CompositeDiffIK::NullspaceTargetPtr n3(new
VirtualRobot::CompositeDiffIK::NullspaceTarget(robotNodeSet, mmm_visualized-
>getRobotNode("RKsegment_joint"), simox::math::mat4f_to_pos(rightKneePose_RootFrame)));
        n3->kP = 0.3;
        cik.addNullspaceGradient(n3);


        // Joint Limit Avoidance in Nullspace - Adapt joint angles in nullspace to try to prevent
values near joint angles
        VirtualRobot::CompositeDiffIK::NullspaceJointLimitAvoidancePtr nsjla(new
VirtualRobot::CompositeDiffIK::NullspaceJointLimitAvoidance(robotNodeSet));
        nsjla->kP = 0.1;
        cik.addNullspaceGradient(nsjla);


        VirtualRobot::CompositeDiffIK::Parameters cp;
        if (std::abs(currentTimestep - timestep) > 0.035) {
            // If time change is large, do {cp.steps} of IK iterations
            cp.steps = 200;
        }
        else {
            // If time change is small, change in joint angles should be small, therefore only do 1
IK step with small change in joint angles
            cp.steps = 1;
            cp.maxJointAngleStep = 0.01;
        }
        cp.resetRnsValues = false;
        cp.returnIKSteps = true;


        auto result = cik.solve(cp);
        MMM_INFO << "Reached " << result.reached << std::endl;
        // Solved joint angles can be retrieved from mmm_visualized if required for storing motions
        this->currentTimestep = timestep;
    }
}
```

```
void MotionAdaptationToolV2HandlerDialog::open(MMM::MotionRecordingPtr motions) {
    this->motions = motions->clone(); // clone motions so it can be adapted while preserving the
original motion
    loadMotions();
}


void MotionAdaptationToolV2HandlerDialog::loadMotions() {
    currentTimestep = -1.0f;
    mmm = nullptr;
    objects.clear();
    SoSeparator* visualization = new SoSeparator();
    SoUnits *u = new SoUnits();
    u->units = SoUnits::MILLIMETERS;
    visualization->addChild(u);
    Eigen::Matrix4f transformation = Eigen::Matrix4f::Identity();
    transformation(1,3) = 2000; // move motion to the left for visualization purpose
    SoMatrixTransform* mt = new SoMatrixTransform();
    SbMatrix m_(reinterpret_cast<SbMat*>(transformation.data()));
    mt->matrix.setValue(m_);
    visualization->addChild(mt);
    for (const std::string &name : motions->getMotionNames()) {
        auto motion = motions->getMotion(name);
        auto robot =  motion->getModel()->cloneScaling();
        if (robot) {
            if (motion->isReferenceModelMotion()) {
                mmm_visualized = robot; // robot model for visualization
                mmm = motion->getModel()->cloneScaling(); // create a second robot model for ik
solving
            }
            else {
                objects[motion->getName()] = robot;
                ui->ChooseMotionComboBox->addItem(QString::fromStdString(motion->getName())); // add
object names to combo box gui to choose from
                if (!currentObject) currentObject = motion;
            }
            robot->reloadVisualizationFromXML();
            robot->setupVisualization(true, true);
            std::shared_ptr<VirtualRobot::CoinVisualization> robot_vis = robot-
>getVisualization<VirtualRobot::CoinVisualization>(VirtualRobot::SceneObject::Full);


            if (!motion->isReferenceModelMotion()) {
                // If Object e.g. create point cloud from robot_vis
            }


            visualization->addChild(robot_vis->getCoinVisualization());
        }
    };
    // TODO Here you can create a new representation for you motion in the frames of objects/the
environment
    // ...


    emit addVisualisation(visualization); // emit a qsignal to update visualization after motion is
changed
}


void MotionAdaptationToolV2HandlerDialog::setCurrentMotion(const QString &name) {
    currentObject = motions->getMotion(name.toStdString()); // sets the current object from gui
}


void MotionAdaptationToolV2HandlerDialog::storeMotion() {
    std::filesystem::path defaultFilePath = std::string(motions->getOriginFilePath().parent_path())
+ std::string(motions->getOriginFilePath().stem()) + "_adapted.xml";
    std::filesystem::path motionFilePath = QFileDialog::getSaveFileName(this, tr("Save motions"),
QString::fromStdString(defaultFilePath), tr("XML files (*.xml)")).toStdString();
    if (!motionFilePath.empty()) {
        motions->saveXML(motionFilePath, true);
        // TODO Currently saves the initial motion - Run your solution over all timesteps and create
new motions to store
    }
}


SoSeparator*
MotionAdaptationToolV2HandlerDialog::createPointCloudVisualization(std::shared_ptr<VirtualRobot::Coi
nVisualization> visualization, int samples, float pointSize) {
    SoSeparator* allSep = new SoSeparator();
    // Insert color information into scene graph
```

```
    for (auto node : visualization->getVisualizationNodes()) {
        if (!node) continue;
        auto trimeshModel = node->getTriMeshModel();

        // following method can be used to sample a point cloud in local coordinates
        // the other part is just for visualization
        std::vector<Eigen::Vector3f> vertices =
VirtualRobot::TriMeshUtils::uniform_sampling(trimeshModel, samples);

        // Add point coordinates
        SoSeparator* sep = new SoSeparator();
        SoCoordinate3* coordinates = new SoCoordinate3();
        std::vector<SbVec3f> pointData;
        pointData.reserve(pointSize);
        for (int i = 0; i < pointSize; i++)
        {
            SbVec3f pointContainer;
            pointContainer[0] = vertices[i](0);
            pointContainer[1] = vertices[i](1);
            pointContainer[2] = vertices[i](2);

            pointData.push_back(pointContainer);
        }
        coordinates->point.setValues(0, pointData.size(), pointData.data());
        sep->addChild(coordinates);

        sep->addChild(VirtualRobot::CoinVisualizationFactory::getMatrixTransform(node-
>getGlobalPose()));

        // Set point size
        SoDrawStyle* sopointSize = new SoDrawStyle();
        sopointSize->pointSize = pointSize;
        sep->addChild(sopointSize);

        // Draw a point set out of all that data
        SoPointSet* pointSet = new SoPointSet();
        sep->addChild(pointSet);
        allSep->addChild(sep);
    }

    return allSep;
}

std::vector<Eigen::Vector3f>
MotionAdaptationToolV2HandlerDialog::createPointCloud(std::shared_ptr<VirtualRobot::CoinVisualizatio
n> visualization, VirtualRobot::RobotPtr robot, int n) {
    std::vector<Eigen::Vector3f> pointCloud;
    int samples = n / visualization->getVisualizationNodes().size();
    for (auto node : visualization->getVisualizationNodes()) {
        if (!node) continue;
        auto trimeshModel = node->getTriMeshModel();
        std::vector<Eigen::Vector3f> vertices =
VirtualRobot::TriMeshUtils::uniform_sampling(trimeshModel, samples);
        Eigen::Matrix4f globalPose = node->getGlobalPose();
        Eigen::Matrix3f rotation = simox::math::mat4f_to_mat3f(globalPose);
        Eigen::Vector3f translation = simox::math::mat4f_to_pos(globalPose);
        // transfer from visualization node to local robot coordinate system
        for (const Eigen::Vector3f vertex : vertices) {
            // transfer to global pose
            Eigen::Vector3f vertex_global = rotation * vertex + translation;
            // transfer to local robot coordinate system
            pointCloud.push_back(robot->toGlobalCoordinateSystemVec(vertex_global));
        }
    }
    return pointCloud;
}
```