

Kinect Object- Motion & PoseDetection

Verwendete Bibliotheken

OpenCV Version 2.4.9

Für Objektdetektion, Berechnungen von Vektoren und Winkeln sowie Visualisierung des Farbbildes.

OpenNI 2 *Open Natural Interaction*

Für Erfassung von Farb- und Tiefendaten des Kinect

Nite 2

Zur Erfassung von Benutzerdaten (verschiedenen Körperteilen)

Programmablauf:

Initialisierung

```
int main(int argc, char** argv)
{
    openni::Status rc = openni::STATUS_OK;

    rc = Init(argc, argv);
    if (rc != openni::STATUS_OK)
    {
        while (1)
        {
            int key = waitKey(30);
            if (key == 27);
            {
                ShutDown();
            }

        }
        return 1;
    }

    Run();
    return 0;
}
```

➔ Einstiegspunkt des Programms, Funktion Init wird aufgerufen.

Zugehöriger Header:

```
Device device;
openni::VideoStream depth, color;
openni::VideoStream** stream;
openni::VideoFrameRef depthFrame, colorFrame;
nite::UserTracker userTracker;
nite::Status niteRc;
nite::UserTrackerFrameRef userTrackerFrame;
openni::Status Init(int argc, char** argv);
openni::Status InitOpenGL(int argc, char** argv);
```

```
openni::Status Init(int argc, char** argv)
{
    NiTE::initialize();
    OpenNI::initialize();
```

```

    if (device.open(openni::ANY_DEVICE) != 0)
    {
        std::cout << "Kinect not found !" << endl;
        openni::OpenNI::getExtendedError();
        std::cout << "Press ESC to exit" << endl;
    }

    std::cout << "Kinect opened" << endl;

    color.create(device, SENSOR_COLOR);
    color.start();
    std::cout << "Camera ok" << endl;
    depth.create(device, SENSOR_DEPTH);
    depth.start();
    std::cout << "Depth sensor ok" << endl;
    openni::VideoMode video;
    video.setResolution(640, 480);
    video.setFps(30);
    video.setPixelFormat(PIXEL_FORMAT_DEPTH_100_UM);
    depth.setVideoMode(video);
    video.setPixelFormat(PIXEL_FORMAT_RGB888);
    color.setVideoMode(video);

    niteRc = userTracker.create(&device);

    if (niteRc != nite::STATUS_OK)
    {
        printf("Couldn't create user tracker\n");
    }
    return InitOpenGL( argc, argv);
}

```

Erklärung

- `NiTE::initialize();`
`OpenNI::initialize();`
 - ➔ Initialisierung der OpenNI und NiTE APIs
- `device.open(openni::ANY_DEVICE)`
 - ➔ Öffnet den Kinectsensor und initialisiert ihn als vorher Deklariertes Objekt vom Typ `openni::Device`. Das Objekt besitzt einen vorgegeben Konstruktor. Mit Hilfe dieses Objektes können verschiedene Informationen wie Tiefen- oder Farbstreams vom Kinect abgerufen werden.
- `color.create(device, SENSOR_COLOR);`
`color.start();`
`depth.create(device, SENSOR_DEPTH);`
`depth.start();`
 - ➔ Im Programmheader deklarierte Objekte (`openni::VideoStream depth, color;`) vom Typ `VideoStream` isolieren einzelne Streams vom Kinect. Mit dem Schlüsselwort `.create` wird ein solcher Stream von einem bestimmten verfügbaren Device mit einem bestimmten Typ initialisiert.
 - ➔ `.start` initialisiert den Datenfluss eines Streams Vom Kinect zum Videostream. Von dem im *Mainloop* nachher einzelne Bilder entnommen werden.

- ```
openni::VideoMode video;
video.setResolution(640, 480);
video.setFps(30);
video.setPixelFormat(PIXEL_FORMAT_DEPTH_100_UM);
depth.setVideoMode(video);
video.setPixelFormat(PIXEL_FORMAT_RGB888);
color.setVideoMode(video);
```

➔ Über die Klasse VideoMode werden hier verschiedene Einstellungen für die Videostreams vorgenommen damit die richtigen Daten entnommen werden können.
- ```
nite::UserTracker userTracker;
niteRc = userTracker.create(&device);
if (niteRc != nite::STATUS_OK)
{
    printf("Couldn't create user tracker\n");
}
```

➔ Der *UserTracker* ist das Hauptobjekt des UserTracker Algorithmus. Er stellt die Hälfte aller Algorithmen zur Verfügung die NiTE anbietet, wie z.B. das Skeletontracking bzw. bestimmte Gelenke und bestimmte Posendetektierung.
.create initialisiert diesen Usertracker und liefert einen Status zurück über den Erfolg der Initialisierung.

```
openni::Status InitOpenGL(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(1280, 680);

    cvNamedWindow("Camera", CV_WINDOW_AUTOSIZE);
    cvNamedWindow("Thresholded Image", CV_WINDOW_AUTOSIZE);
    createTrackbars();
    glutCreateWindow("Kinect 3DSkeleton");
    glEnable(GL_DEPTH_TEST);

    glutReshapeFunc(changeSize);
    glutKeyboardFunc(processNormalKeys);
    glutSpecialFunc(handleKeypress);

    glutDisplayFunc(MainLoop);
    glutIdleFunc(myIdle);

    return openni::STATUS_OK;
}
void Run()
{
    glutMainLoop();
}

void myIdle()
{
    glutPostRedisplay();
}
```

- ➔ Initialisierung und Einstellung von OpenGL zur Visualisierung. Und Erstellung von OpenCV Fenstern

Der Mainloop

```
void MainLoop()
```

Dies ist die „Hauptfunktion“ des Programms sie wird von OpenGL immer wieder automatisch aufgerufen. Hier werden einzelne Frames aus den Streams gelesen und es finden die Auswertungen der Userdaten statt. Zudem wird hier auf weitere Funktionen verwiesen die zur Visualisierung, Berechnung von Abständen und Winkeln sowie Objektdetektion dienen.

Ich werde diese Funktion nun auch Schritt für Schritt erklären und an entsprechenden Stellen auf aufgerufene Funktionen eingehen ... Ich hoffe dies ist verständlich :D

- `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glClearColor(0, 0, 0, 0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();`

```
//camera  
gluLookAt(x, -2.0f, z,  
          x + lx, -2.0f, z + lz,  
          0.0f, 1.0f, 0.0f);
```

- ➔ Hier wird das Fenster von OpenGL geleert und entsprechende Einstellungen zur Darstellung getroffen. Durch `gluLookAt` wird die Position und Blickrichtung der virtuellen Kamera eingestellt. Diese können auch durch entsprechende Tastendrücke verändert werden.
- `color.readFrame(&colorFrame);
depth.readFrame(&depthFrame);
cameraFeed.data = (uchar*)colorFrame.getData();
Objecttracking();`

➔ Hier wird aus dem Farb- und TiefenStream jeweils ein einzelner Frame aufgenommen. Diese werden in einem Objekt vom Typ `openni::VideoFrameRef` gespeichert. Die Daten des `colorFrame` werden in ein Objekt vom Typ `cv::ArrMat` übertragen. Auf Grundlage dieses Images werden die Berechnungen für das Objecttracking gemacht.

➔ Anschließend wird die Funktion `Objecttracking()` aufgerufen, in welcher die Position des Objektes anhand der Farbe detektiert wird. Ich erläutere diesen Algorithmus im Folgenden.

Das Objecttracking

Code:

Aus dem Header:

```
float posX = 0;
float posY = 0;
Point2f Object = { 0, 0 };
Mat cameraFeed(cv::Size(640, 480), CV_8UC3, NULL);
Mat HSV(cv::Size(640, 480), CV_8UC3, NULL);
Mat thresholdMat(cv::Size(640, 480), CV_8UC1, NULL);
cv::Mat depthcv1(cv::Size(640, 480), CV_16UC1, NULL);
IplImage* Contours = cvCreateImage(cvSize(640, 480), IPL_DEPTH_8U, 1);
IplImage* drawingIpl = cvCreateImage(cvSize(640, 480), IPL_DEPTH_8U, 1);
```

Funktion:

```
void Objecttracking()
{
    Mat temp;
    cvSet(drawingIpl, cvScalar(0));
    CvMoments *colorMoment = (CvMoments*)malloc(sizeof(CvMoments));
    static const int thickness = 3;
    static const int lineType = 8;
    Scalar color = CV_RGB(255, 255, 255);

    temp = thresholdMat.clone();
    Contours->imageData = (char*)temp.data;

    CvMemStorage* storage = cvCreateMemStorage(0);
    CvSeq* contours = 0;
    CvSeq* biggestContour = 0;
    int numCont = 0;
    int contAthresh = 45;

    cvFindContours(Contours, storage, &contours, sizeof(CvContour),
                  CV_RETR_LIST, CV_CHAIN_APPROX_SIMPLE, cvPoint(0, 0));

    int largest_area = 0;

    for (; contours != 0; contours = contours->h_next)
    {
        double a = cvContourArea(contours, CV_WHOLE_SEQ);

        if (a > largest_area){

            largest_area = a;
            biggestContour = contours;
        }
    }

    if (biggestContour != 0)
    {
        cvDrawContours(drawingIpl, biggestContour, color, color, -1,
                      thickness, lineType, cvPoint(0, 0));
    }

    cvMoments(drawingIpl, colorMoment, 1);
    if (enableObjectTracking)
    {
```

```

        double moment10 = cvGetSpatialMoment(colorMoment, 1, 0);
        double moment01 = cvGetSpatialMoment(colorMoment, 0, 1);
        double area = cvGetCentralMoment(colorMoment, 0, 0);
        if (area > 30)
        {

            posX = (moment10 / area);
            posY = moment01 / area;
            Object.x = posX;
            Object.y = posY;

            objectTracked = true;
            //int depthIndex = posX + (posY * 640);
            string string = "Tracking Object ";
            putText(cameraFeed, string, Point(30, 450), 1, 1, Scalar(0, 255, 0), 2);
            drawObject(posX, posY, cameraFeed);
        }

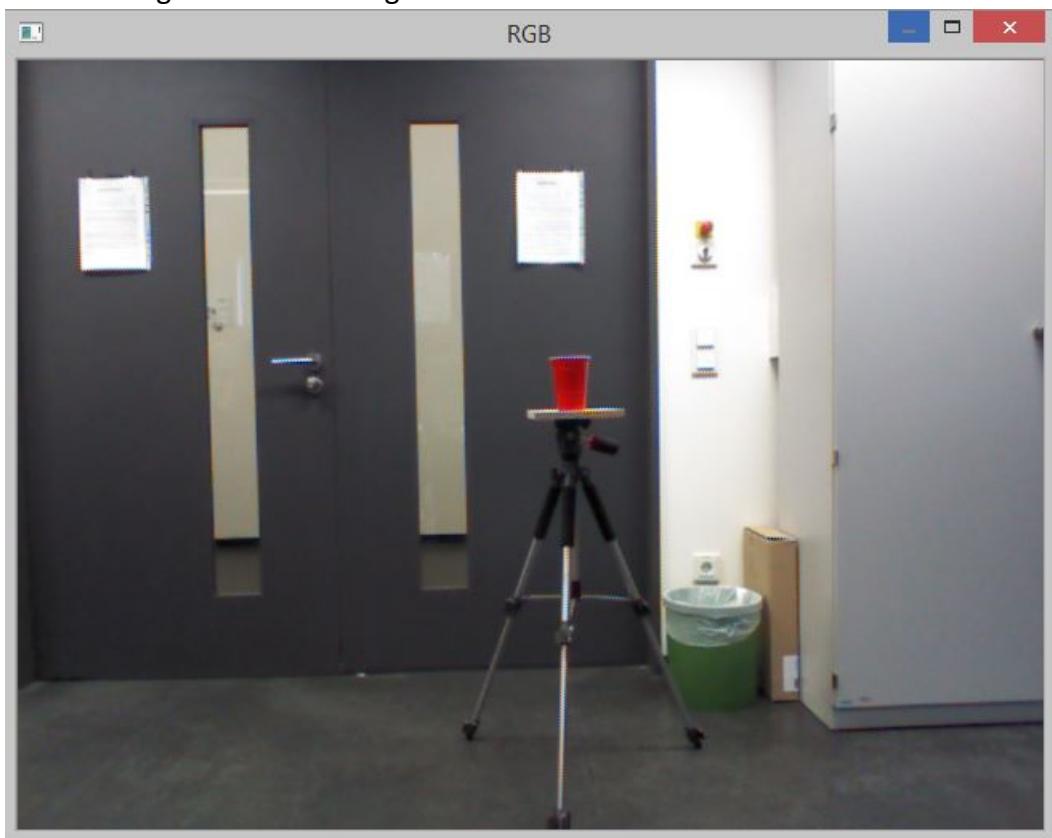
        else
        {
            objectTracked = false;
            putText(cameraFeed, "No Object", Point(30, 450), 1, 1, Scalar(0, 255, 0), 2);
        }
    }

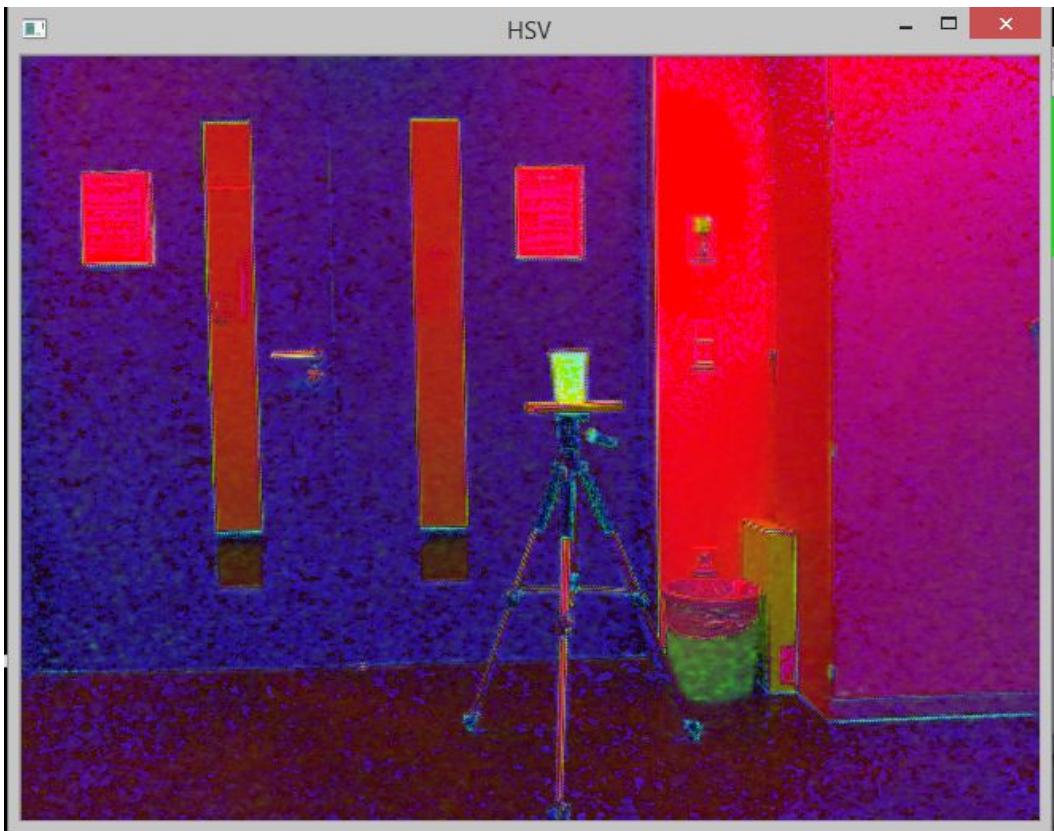
    else
    {
        objectTracked = false;
    }
}

```

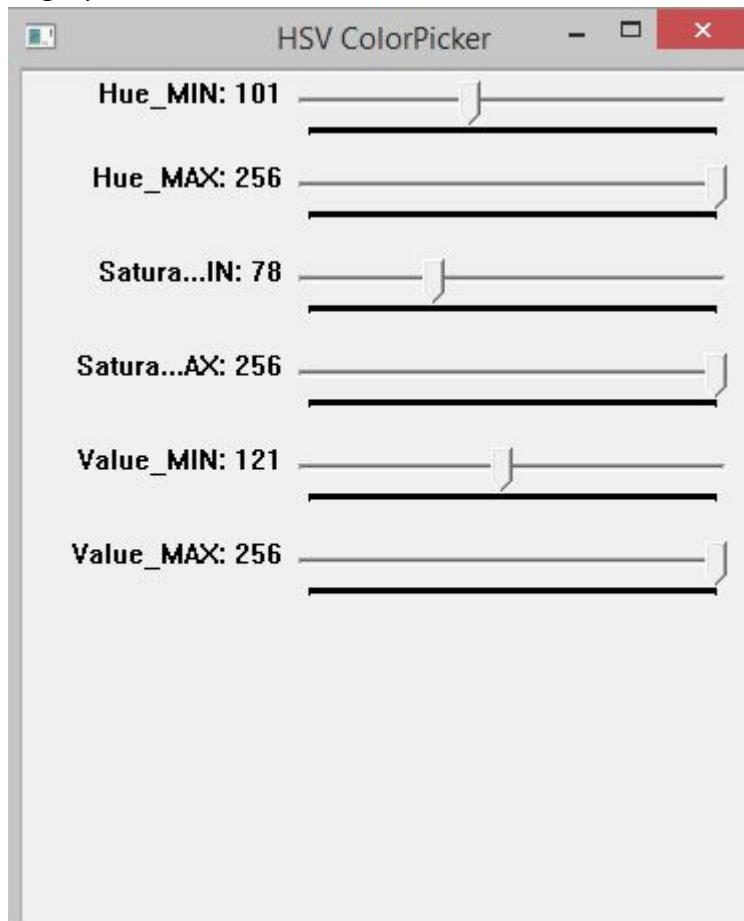
Codeerklärung:

- `cvtColor(cameraFeed, HSV, COLOR_BGR2HSV)`
- ➔ von RGB in HSV Farbraum. Für Farbdetektion besser, da auch Sättigung und Helligkeit berücksichtigt werden.





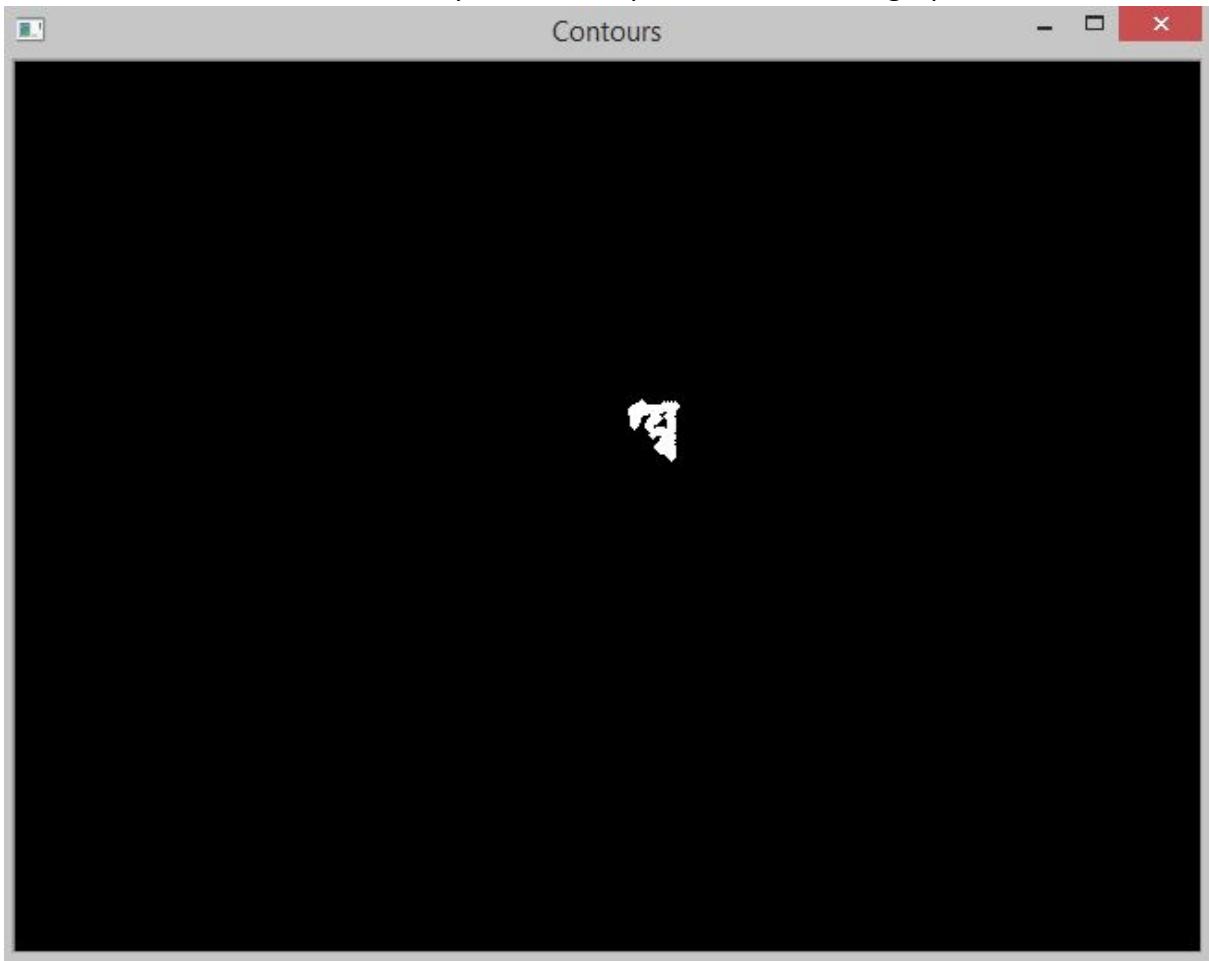
- `cv::inRange(HSV, Scalar(H_MIN, S_MIN, V_MIN), Scalar(H_MAX, S_MAX, V_MAX), thresholdMat);`
→ Prüfen welche Pixel sich in indem ausgewählten Farbraum befinden.
Entsprechende Pixel werden in dem 1 Channel image *thresholdMat* abgespeichert.





- Mat temp;cvSet(drawingIpl, cvScalar(0));*temp = thresholdMat.clone();*
Contours->imageData = (char)temp.data;*
→ Konvertierung zum IPL ImageFormat (Hinweis: Dies habe ich nur aus dem Grund getan, da die Folgenden FindContours Funktion mit Vektoren als Output im Mat Format bei mir zu Fehlern geführt hat.

- `cvFindContours(Contours, storage, &contours, sizeof(CvContour), CV_RETR_LIST, CV_CHAIN_APPROX_SIMPLE, cvPoint(0, 0));`
 ➔ Das Image *Contours* wird auf vorhandene Konturen überprüft. Entsprechende Konturen werden in dem dynamischen Speicher *contours* abgespeichert



- `int largest_area = 0;`
`for (; contours != 0; contours = contours->h_next)`
`{double a = cvContourArea(contours, CV_WHOLE_SEQ);`
`if (a > largest_area){`
`largest_area = a;`
`biggestContour = contours;}`
`}`
 ➔ In dieser Schleife wird für jede gefundene Kontur in *Contours* die Anzahl der Pixel ausgewertet und die pixelmäßig größte Kontur gefunden.
- `if (biggestContour != 0)`
`{cvDrawContours(drawingIpl, biggestContour, color, color, -1, thickness, lineType,`
`cvPoint(0, 0));}`
 ➔ Die größte gefundene Kontur wird in einen one Channel IPL image als weiße Pixel eingzeichnet/gespeichert.
- `cvMoments(drawingIpl, colorMoment, 1);`

```

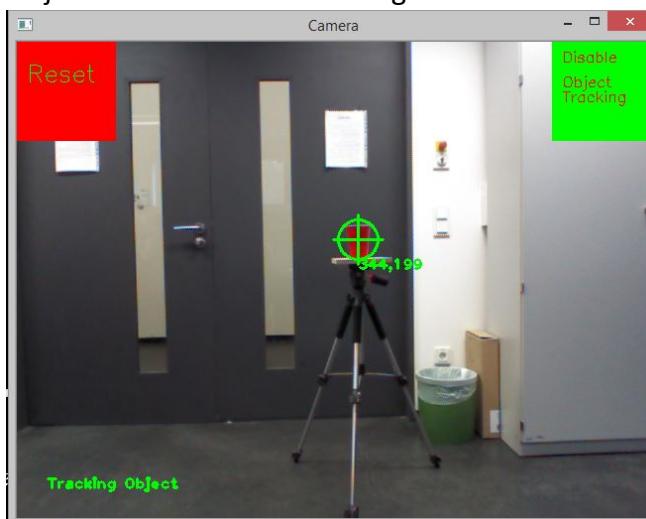
double moment10 = cvGetSpatialMoment(colorMoment, 1, 0);
double moment01 = cvGetSpatialMoment(colorMoment, 0, 1);
double area = cvGetCentralMoment(colorMoment, 0, 0);
➔ in moment10 wird die Summe aller x-Koordinaten der weißen Pixel abgespeichert,
in moment01 entsprechend der y-Koordinaten. In area die Summe aller weißen
Pixel.

```

- ```

if (area > 30){
 posX = (moment10 / area);
 posY = moment01 / area;
 Object.x = posX;
 Object.y = posY;
 objectTracked = true;
 string string = "Tracking Object ";
 putText(cameraFeed, string, Point(30, 450), 1, 1, Scalar(0, 255,
 0), 2);
 drawObject(posX, posY, cameraFeed);}
else{
 objectTracked = false;
 putText(cameraFeed, "No Object", Point(30, 450), 1, 1, Scalar(0, 255, 0), 2);}
➔ Für x- und y-Koordinaten wird ein Mittelwert gebildet und als globale Variable
gespeichert. Ein globaler bool objectTracked wird zu 1 gesetzt um anderen
Funktionen zu sagen, dass ein Objekt detektiert wurde. Die Koordinate des
Objektes wird in das Farbbild gezeichnet.

```



Zurück zum MainLoop....

Wenn sie in den Programmcode gucken folgen nun einige ereignisorientierte Anzeigen im Hauptfenster Visualisieren. Diese sind aber eher nebensächlich.

Der Hauptalgorithmus geht mit folgendem Befehl weiter:

- ```

nite::Status niteRc1;
niteRc1 = userTracker.readFrame(&userTrackerFrame);
if (niteRc1 != nite::STATUS_OK)
{
    printf("Couldn't read Frame");
}
➔ Hier wird ein einzelner Frame vom Usertracker gelesen. Und als Objekt vom Typ
UserTrackerFrameRef abgespeichert. Dieses Objekt ist prinzipiell ein Image vom

```

Depthframe. Jedoch enthält es zusätzlich alle Informationen zu jedem User der im Bild gefunden wurde, und Informationen zum Boden.

- ```
nite::Point3f Floorpoint;
Floorpoint.y = userTrackerFrame.getFloor().point.y;

if (userTrackerFrame.getFloorConfidence() > 0.8f)
{
 DrawButton(Floorpoint.y);
}
else
{
 DrawButton(); // Boden mit fixem Wert Zeichnen
}
```

- ➔ Nun wird aus dem UserTrackerframe die Information über die y (also die Vertikale Koordinate ) des detektierten Boden eingelesen. Diese ist in Realworld Koordinaten also die Distanz mm vom Sensor.
- ➔ Die Funktion .getFloorCondience gibt einen Wert zwischen 0 und 1 zurück, der die Sicherheit der Berechnung angibt. Also die Wahrscheinlichkeit, dass sich benanntes Objekt ach dort befindet.
- ➔ Ist diese Sicherheit größer 0,8 wird Die Funktion Drawbutton mit der Vertikalen Koordinate des Bodens aufgerufen. So wird sichergestellt, dass der Boden je nach Lage des KinectSensors immer in der richtigen Höhe eingezeichnet wird.

- ```
const nite::Array<nite::UserData>& users = userTrackerFrame getUsers();
```

 - ➔ *nite::Array* : Simples Klasse der Nite API zur Erstellung eines Arrays bestimmter Objekte.
 - ➔ *Nite::Usedata* stellt alle Informationen eines spezifischen Users, der von dem UserTracker detektiert wurde.

- ➔ Ganz konkret wird hier ein Array mit dem Namen *users* (im Programmheader deklariert) erststellt, welches durch den Befehl .getUsers() mit den Userdaten aller im Bild detektierten User initialisiert wird.

- ```
for (int i = 0; i < users.getSize(); ++i)
{
 const nite::UserData& user = users[i];
```

  - ➔ Hier startet eine Schleife innerhalb des *MainLoops*. Diese Schleife wird für jeden detektierten User innerhalb des *MainLoops* einmal ausgeführt. Und enthält, wie im Folgenden gezeigt, alle Funktionen Für die Visualisierung und die Posendetektierung. Alles was nun folgt wird für jeden User einmal ausgeführt.
  - ➔ **Hinweis:** Im jetzigen Status des Projektes wird zwar jeder User visualisiert, jedoch ist die Posendetektierung nur für einen User ausgelegt. Soll heißen, dass wenn sich mehrere User im Bild befinden, die Posendetektierung für alle funktioniert, jedoch entsprechende Winkel nur für einen angezeigt werden können.

- ```

        if (user.isNew())
            std::cout << "New" << endl;
        else if (user.isVisible() && !g_visibleUsers[user.getId()])
            renderBitmapString(150, 80, (void *)font, "User Visible", 3);
        else if (!user.isVisible() && g_visibleUsers[user.getId()])
        {

            renderBitmapString(150, 80, (void *)font, "No User ", 1);

        }
        else if (user.isLost())
            std::cout << "Lost" << endl;

        g_visibleUsers[user.getId()] = user.isVisible();

        if (g_skeletonStates[user.getId()] != user.getSkeleton().getState())
        {
            switch (g_skeletonStates[user.getId()] =
user.getSkeleton().getState())
            {
                case nite::SKELETON_NONE:
                    std::cout << "Stopped tracking." << endl;

                    break;
                case nite::SKELETON_CALIBRATING:
                    std::cout << "Calibrating..." << endl;

                    break;
                case nite::SKELETON_TRACKED:
                    std::cout << "Tracking!" << endl;

                    break;
                case nite::SKELETON_CALIBRATION_ERROR_NOT_IN_POSE:
                case nite::SKELETON_CALIBRATION_ERROR_HANDS:
                case nite::SKELETON_CALIBRATION_ERROR_LEGS:
                case nite::SKELETON_CALIBRATION_ERROR_HEAD:
                case nite::SKELETON_CALIBRATION_ERROR_TORSO:
                    std::cout << "Calibration Failed... " << endl;

                    break;
            }
        }
    }

```

- Eine Reihe von Abfragen zum Status des Users und Rückmeldung in der Konsole.
Relativ selbsterklärend, und für den eigentlichen Algorithmus eher uninteressant.

- ```

 if (user.isNew())
 {
 userTracker.startSkeletonTracking(user.getId());
 }

```
- Wenn der User neu im Bild erscheint soll für ihn das SkeletonTracking gestartet werden.

```

 else if (user.getSkeleton().getState() == nite::SKELETON_TRACKED)
 {
 renderBitmapString(600, 50, (void *)font, "User Visible", 3);
 if (objectTracked)
 {
 cv::Point2f RightHand1;
 userTracker.convertJointCoordinatesToDepth(user.getSkeleton()
().getJoint(nite::JOINT_RIGHT_HAND).getPosition().x,
user.getSkeleton().getJoint(nite::JOINT_RIGHT_HAND).getPosition().y,
user.getSkeleton().getJoint(nite::JOINT_RIGHT_HAND).getPosition().z,
&RightHand1.x, &RightHand1.y);

```

```

cv::Point2f LeftHand1;

userTracker.convertJointCoordinatesToDepth(user.getSkeleton().getJoint(nite::JOINT_LEFT_HAND).getPosition().x,
 user.getSkeleton().getJoint(nite::JOINT_LEFT_HAND).getPosition().y,
 user.getSkeleton().getJoint(nite::JOINT_LEFT_HAND).getPosition().z,
 &LeftHand1.x, &LeftHand1.y);

cv::Vec2f Object_to_righthand((RightHand1.x - Object.x),
 (RightHand1.y - Object.y));
cv::Vec2f Object_to_lefthand((LeftHand1.x - Object.x),
 (LeftHand1.y - Object.y));

double distance_Object_right_Hand =
sqrt(pow(Object_to_righthand[0], 2) + pow(Object_to_righthand[1], 2));
double distance_Object_left_Hand =
sqrt(pow(Object_to_lefthand[0], 2) + pow(Object_to_lefthand[1], 2));

if (distance_Object_right_Hand < 50.0f){
 contact_right = true;
}
else{
{
 contact_right = false;
}

if (distance_Object_left_Hand < 50.0f{
 contact_left = true; }
else{
 contact_left = false; }
}
else{
 contact_right = false;
 contact_left = false;
}

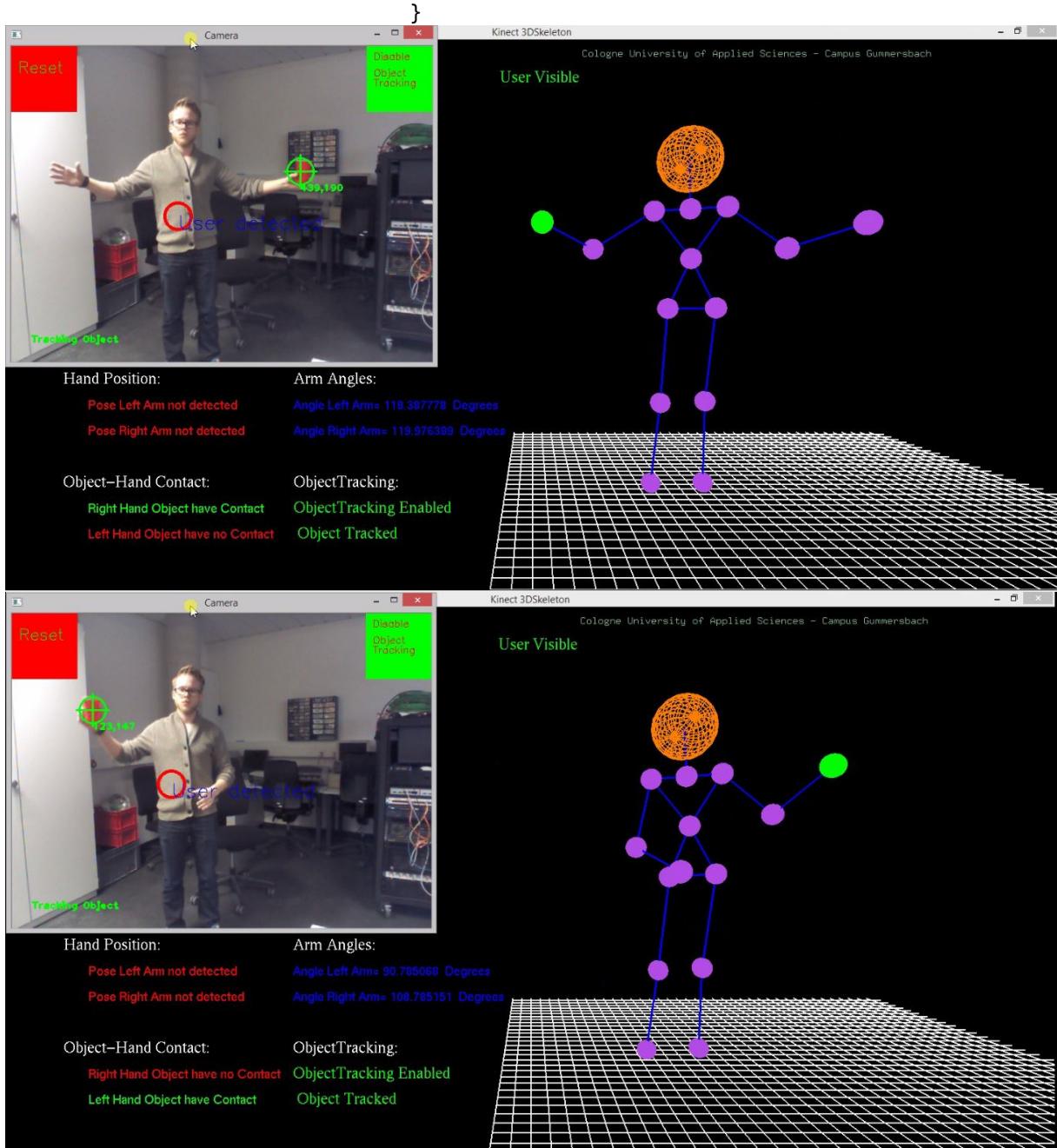
```

- In diesen Algorithmus, wird zuerst abgefragt, ob das Skeleton des Users verfolgt wird. Ist dies der Fall, so wird die 3D Koordinate der Rechten und linken Hand berechnet(.getSkeleton().getJoint()) und in 2D Pixelkoordinaten umgerechnet(convertJointCoordinatesToDepth()).
- Nun wird die 2 dimensionalen Vektoren zwischen linker bzw. rechter Hand und dem in Echtzeit detektierten Objekt berechnet. cv::Vec2f Object\_to\_righthand((RightHand1.x - Object.x), (RightHand1.y - Object.y));
- Von diesen Vektoren wird der Betrag also die Länge des Vektors in Pixel berechnet. Diese ist wohl allgemein bekannt als die Quadratwurzel der Summe aller Vektorkomponenten zum Quadrat.  
`double distance_Object_right_Hand = sqrt(pow(Object_to_righthand[0], 2) + pow(Object_to_righthand[1], 2));`
  
- Ist nun diese Länge kleiner als ein vorgegebener Wert, so wird dem gesamten Programm gesagt, dass ein Kontakt zwischen rechter oder linker Hand und Objekt stattfindet, indem entsprechend globale boolsche Ausdrücke zu 1 gesetzt werden. Entsprechende Hand wird grün visualisiert.

```

if (distance_Object_right_Hand < 50.0f){
 contact_right = true;
}
else
{
 contact_right = false;
}

```



- ```

        DrawLimb(user.getSkeleton().getJoint(nite::JOINT_HEAD),
        user.getSkeleton().getJoint(nite::JOINT_NECK));

        DrawLimb(user.getSkeleton().getJoint(nite::JOINT_LEFT_SHOULDER),
        user.getSkeleton().getJoint(nite::JOINT_LEFT_ELBOW));
        DrawLimb(user.getSkeleton().getJoint(nite::JOINT_LEFT_ELBOW),
        user.getSkeleton().getJoint(nite::JOINT_LEFT_HAND));

        DrawLimb(user.getSkeleton().getJoint(nite::JOINT_RIGHT_SHOULDER),
        user.getSkeleton().getJoint(nite::JOINT_RIGHT_ELBOW));
        DrawLimb(user.getSkeleton().getJoint(nite::JOINT_RIGHT_ELBOW),
        user.getSkeleton().getJoint(nite::JOINT_RIGHT_HAND));

        DrawLimb(user.getSkeleton().getJoint(nite::JOINT_LEFT_SHOULDER),
        user.getSkeleton().getJoint(nite::JOINT_RIGHT_SHOULDER));
    
```

```

        DrawLimb(user.getSkeleton().getJoint(nite::JOINT_LEFT_SHOULDER),
user.getSkeleton().getJoint(nite::JOINT_TORSO));
        DrawLimb(user.getSkeleton().getJoint(nite::JOINT_RIGHT_SHOULDER),
user.getSkeleton().getJoint(nite::JOINT_TORSO));

        DrawLimb(user.getSkeleton().getJoint(nite::JOINT_TORSO),
user.getSkeleton().getJoint(nite::JOINT_LEFT_HIP));
        DrawLimb(user.getSkeleton().getJoint(nite::JOINT_TORSO),
user.getSkeleton().getJoint(nite::JOINT_RIGHT_HIP));

        DrawLimb(user.getSkeleton().getJoint(nite::JOINT_LEFT_HIP),
user.getSkeleton().getJoint(nite::JOINT_LEFT_KNEE));
        DrawLimb(user.getSkeleton().getJoint(nite::JOINT_LEFT_KNEE),
user.getSkeleton().getJoint(nite::JOINT_LEFT_FOOT));

        DrawLimb(user.getSkeleton().getJoint(nite::JOINT_RIGHT_HIP),
user.getSkeleton().getJoint(nite::JOINT_RIGHT_KNEE));
        DrawLimb(user.getSkeleton().getJoint(nite::JOINT_RIGHT_KNEE),
user.getSkeleton().getJoint(nite::JOINT_RIGHT_FOOT));



        DrawJoint(user.getSkeleton().getJoint(nite::JOINT_NECK));
        DrawJoint(user.getSkeleton().getJoint(nite::JOINT_LEFT_SHOULDER));

DrawJoint(user.getSkeleton().getJoint(nite::JOINT_RIGHT_SHOULDER));
        DrawJoint(user.getSkeleton().getJoint(nite::JOINT_LEFT_ELBOW));
        DrawJoint(user.getSkeleton().getJoint(nite::JOINT_RIGHT_ELBOW));
        DrawJoint(user.getSkeleton().getJoint(nite::JOINT_TORSO));
        DrawJoint(user.getSkeleton().getJoint(nite::JOINT_LEFT_HAND));
        DrawJoint(user.getSkeleton().getJoint(nite::JOINT_RIGHT_HAND));
        DrawJoint(user.getSkeleton().getJoint(nite::JOINT_LEFT_HIP));
        DrawJoint(user.getSkeleton().getJoint(nite::JOINT_RIGHT_HIP));
        DrawJoint(user.getSkeleton().getJoint(nite::JOINT_LEFT_KNEE));
        DrawJoint(user.getSkeleton().getJoint(nite::JOINT_RIGHT_KNEE));
        DrawJoint(user.getSkeleton().getJoint(nite::JOINT_LEFT_FOOT));
        DrawJoint(user.getSkeleton().getJoint(nite::JOINT_RIGHT_FOOT));

        DrawHead(user.getSkeleton().getJoint(nite::JOINT_HEAD));

```

- ➔ Hier werden die Funktionen zur 3D Visualisierung des Skeletes aufgerufen. Kopf, Gelenke und Gliedmaße werden mit Hilfe von OpenGL gezeichnet, in dem jeweils die die konkreten Real-World Koordinaten mit übergeben werden.
- ➔ Hier am Beispiel der Gliedmaße:

```

void DrawLimb( const nite::SkeletonJoint &joint1, const nite::SkeletonJoint &joint2)
{
    if (joint1.getPositionConfidence() > .1f &&
joint2.getPositionConfidence() > .1f)
    {
        cv::Point3f limb1;
        limb1.x = -joint1.getPosition().x / 100+7;
        limb1.y = joint1.getPosition().y / 100;
        limb1.z = ((joint1.getPosition().z / 100)* (-1));
        cv::Point3f limb2;
        limb2.x = -joint2.getPosition().x / 100+7;
        limb2.y = joint2.getPosition().y / 100;
        limb2.z = -joint2.getPosition().z / 100;




        glColor3f(0, 0, 1);
        if ((joint1.getType() == nite::JointType::JOINT_LEFT_SHOULDER &&
joint2.getType() == nite::JointType::JOINT_LEFT_ELBOW)

```

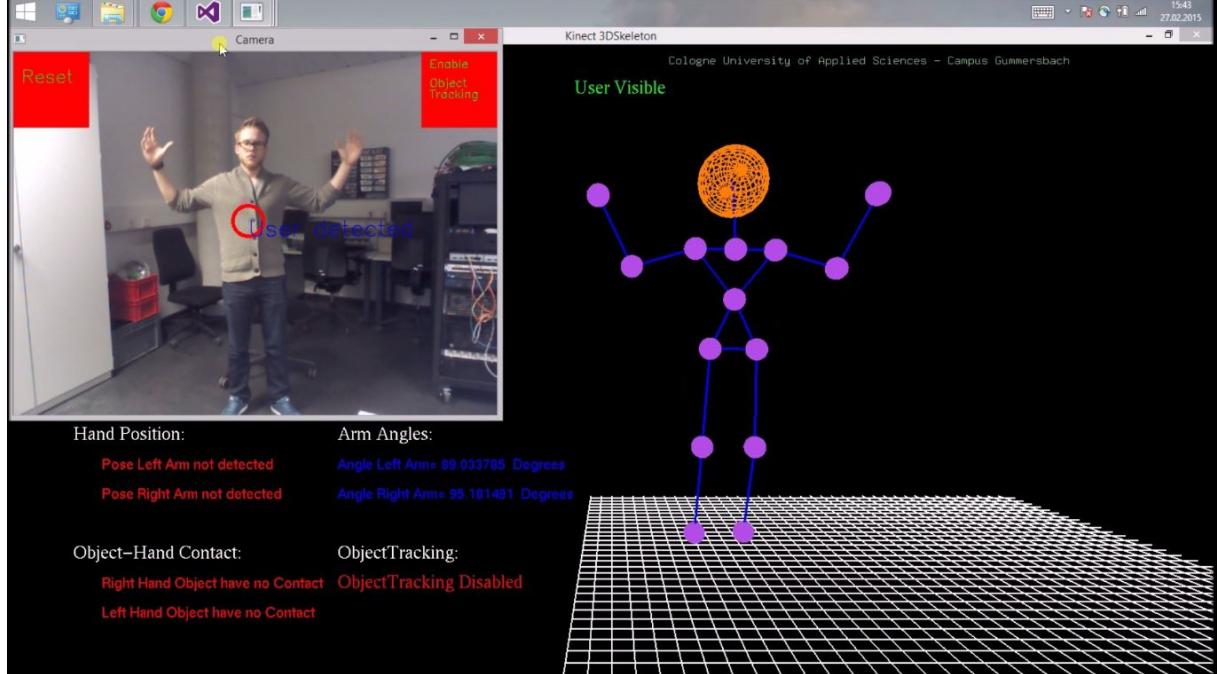
```

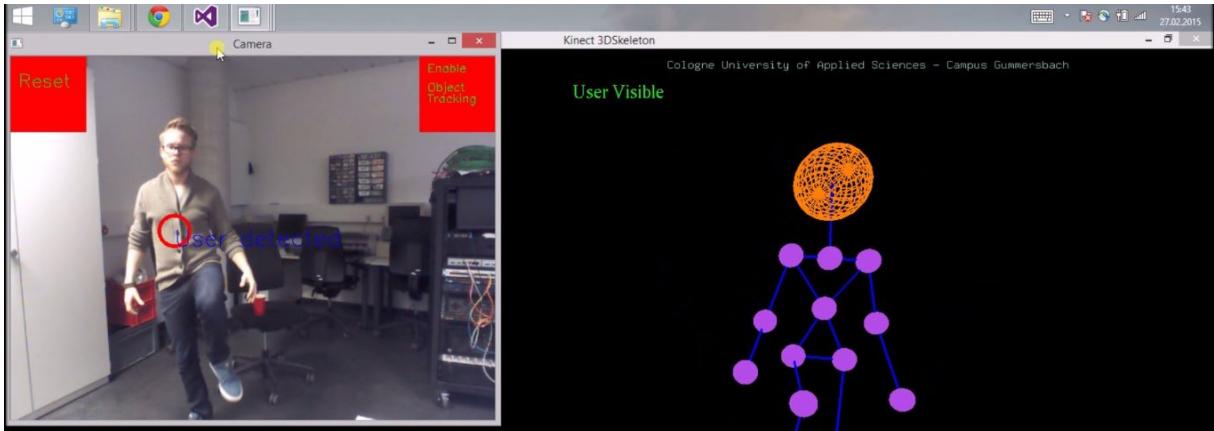
        | (joint1.getType() == nite::JointType::JOINT_LEFT_ELBOW &&
joint2.getType() == nite::JointType::JOINT_LEFT_HAND) &&
(winkel2_degree >= 23.0 && winkel2_degree <= 50.0))
{
    glColor3f(0.0, 1.0, 0.0);
}
        if ((joint1.getType() ==
nite::JointType::JOINT_RIGHT_SHOULDER && joint2.getType() ==
nite::JointType::JOINT_RIGHT_ELBOW)
        | (joint1.getType() ==
nite::JointType::JOINT_RIGHT_ELBOW && joint2.getType() ==
nite::JointType::JOINT_RIGHT_HAND) && (winkel_degree >= 23.0 &&
winkel_degree <= 50.0))
{
    glColor3f(0.0, 1.0, 0.0);
}
glLineWidth(3);
glPushMatrix();
glBegin(GL_LINES);
glTranslatef(0.0f, 0.0f, 0.0f);
glVertex3f(limb1.x, limb1.y, limb1.z);

glVertex3f(limb2.x, limb2.y, limb2.z);

glEnd();
glPopMatrix();
}
}

```





Hand Position:

Pose Left Arm not detected

Pose Right Arm not detected

Arm Angles:

Angle Left Arm= 149.140760 Degrees

Angle Right Arm= 136.821725 Degrees

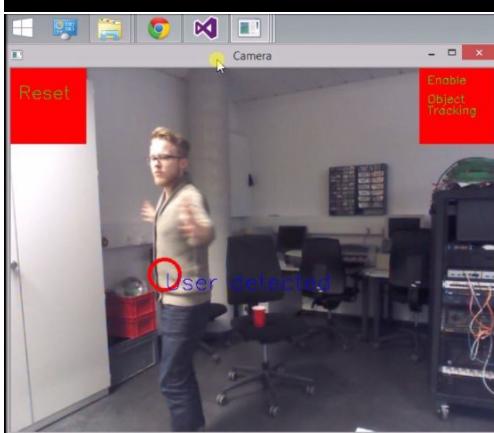
Object-Hand Contact:

Right Hand Object have no Contact

Left Hand Object have no Contact

ObjectTracking:

ObjectTracking Disabled



Hand Position:

Pose Left Arm not detected

Pose Right Arm not detected

Arm Angles:

Angle Left Arm= 136.836248 Degrees

Angle Right Arm= 132.344385 Degree

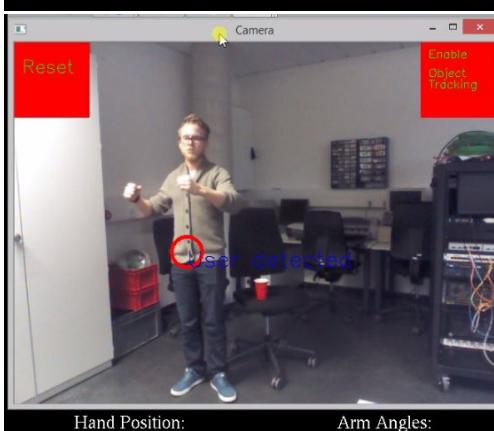
Object-Hand Contact:

Right Hand Object have no Contact

Left Hand Object have no Contact

ObjectTracking:

ObjectTracking Disabled



Hand Position:

Pose Left Arm not detected

Pose Right Arm not detected

Arm Angles:

Angle Left Arm= 60.418060 Degrees

Angle Right Arm= 85.176867 Degrees

Object-Hand Contact:

Right Hand Object have no Contact

Left Hand Object have no Contact

ObjectTracking:

ObjectTracking Disabled



- Zuerst wird die Sicherheit der Koordinatenberechnung geprüft, ist eine bestimmte Sicherheit gegeben, so werden die Koordinaten von mm in dm umgerechnet gleichzeitig die Vektoren zwischen den Einzelnen Gelenken berechnet.
Nun werden diese mit OpenGL dreidimensional visualisiert.
- Zudem werden die Arme wenn ein bestimmter Winkel erreicht ist, grün angezeigt.

- Nun erfolgt die Finale Berechnung der Posendetektierung, hierzu wird die Funktion:
`Angle_and_contact(user.getSkeleton().getJoint(nite::JOINT_RIGHT_SHOULDER),
user.getSkeleton().getJoint(nite::JOINT_RIGHT_ELBOW),
user.getSkeleton().getJoint(nite::JOINT_RIGHT_HAND),
user.getSkeleton().getJoint(nite::JOINT_LEFT_SHOULDER),
user.getSkeleton().getJoint(nite::JOINT_LEFT_ELBOW),
user.getSkeleton().getJoint(nite::JOINT_LEFT_HAND),
user.getSkeleton().getJoint(nite::JOINT_HEAD))`
mit den erforderlichen Koordinaten von Schulter, Elbogen und Hand jeweils für die Rechte und linke Seite, sowie die Koordinaten des Kopfes, alles in real-World Koordinaten.

Die Funktion sieht wie Folgt aus:

```
void Angle_and_contact(const nite::SkeletonJoint &RShoulder, const nite::SkeletonJoint
&RElbow, const nite::SkeletonJoint &RHand,
                      const nite::SkeletonJoint &LShoulder,
                      const nite::SkeletonJoint &LElbow, const nite::SkeletonJoint &LHand,
const nite::SkeletonJoint &Head)
{
    cv::Point3d head(Head.getPosition().x, Head.getPosition().y,
Head.getPosition().z);

    //right
    cv::Point3d r_shoulder(RShoulder.getPosition().x,
RShoulder.getPosition().y, RShoulder.getPosition().z);
    cv::Point3d r_elbow(RElbow.getPosition().x, RElbow.getPosition().y,
RElbow.getPosition().z);
    cv::Point3d r_hand(RHand.getPosition().x, RHand.getPosition().y,
RHand.getPosition().z);

    cv::Vec3d r_Ober(r_elbow.x - r_shoulder.x, r_elbow.y - r_shoulder.y,
r_elbow.z - r_shoulder.z);
    cv::Vec3d r_Unter(r_hand.x - r_elbow.x, r_hand.y - r_elbow.y, r_hand.z -
r_elbow.z);
    cv::Vec3d R_Oberarm = normalize(r_Ober);
    cv::Vec3d R_Unterarm = normalize(r_Unter);

    cv::Vec3d head_to_righthand(r_hand.x - head.x, r_hand.y - head.y,
r_hand.z - head.z);

    double distance_head_right_Hand = sqrt(pow(head_to_righthand[0], 2) +
pow(head_to_righthand[1], 2) + pow(head_to_righthand[2], 2));
    double winkel_radian = acos(R_Oberarm[0] * R_Unterarm[0] + R_Oberarm[1]
* R_Unterarm[1] + R_Oberarm[2] * R_Unterarm[2]);
    winkel_degree = 180 - (180 / M_PI * winkel_radian);

    //left
    cv::Point3d L_shoulder(LShoulder.getPosition().x,
LShoulder.getPosition().y, LShoulder.getPosition().z);
    cv::Point3d L_elbow(LElbow.getPosition().x, LElbow.getPosition().y,
LElbow.getPosition().z);
    cv::Point3d L_hand(LHand.getPosition().x, LHand.getPosition().y,
LHand.getPosition().z);
```

```

        cv::Vec3d L_Ober(L_elbow.x - L_shoulder.x, L_elbow.y - L_shoulder.y,
L_elbow.z - L_shoulder.z);
        cv::Vec3d L_Unter(L_hand.x - L_elbow.x, L_hand.y - L_elbow.y, L_hand.z -
L_elbow.z);
        cv::Vec3d L_Oberarm = normalize(L_Ober);
        cv::Vec3d L_Unterarm = normalize(L_Unter);

        cv::Vec3d head_to_lefthand(L_hand.x - head.x, L_hand.y - head.y, L_hand.z
- head.z);

        double distance_head_left_Hand = sqrt(pow(head_to_lefthand[0], 2) +
pow(head_to_lefthand[1], 2) + pow(head_to_lefthand[2], 2));
        double winkel2_radian = acos(L_Oberarm[0] * L_Unterarm[0] + L_Oberarm[1]
* L_Unterarm[1] + L_Oberarm[2] * L_Unterarm[2]);
        winkel2_degree = 180 - (180 / M_PI * winkel2_radian);

    //Right
    if (RShoulder.getPositionConfidence() > .5f
&& RElbow.getPositionConfidence() > .5f && RHand.getPositionConfidence() > .5f)
    {

        sprintf_s(buffer1, "Angle Right Arm= %f Degrees ",
winkel_degree);
        rAngle = true;

        if (distance_head_right_Hand < 400 &&
Head.getPositionConfidence() > .5f && (RHand.getPositionConfidence() > .5f))

    {

        if (winkel_degree >= 23.0 && winkel_degree <= 50.0)
        {

            rContact = true;
            if (contact_right)
            {
                drink = true;
                rDrinking = true;

            }
            else
            {
                rDrinking = false;
            }
        }
        else
        {
            rDrinking = false;
            rContact = false;
        }

    }
    else
    {
        rDrinking = false;
        rContact = false;
    }

}
else
{
    rDrinking = false;
    rAngle = false;
    rContact = false;

}

//left
if (LShoulder.getPositionConfidence() > .5f &&
LElbow.getPositionConfidence() > .5f && LHand.getPositionConfidence() > .5f)

```

```

    {

winkel2_degree); sprintf_s(buffer2, "Angle Left Arm= %f Degrees ", lAngle = true;

        if (distance_head_left_Hand < 400 && Head.getPositionConfidence() > .5f && (LHand.getPositionConfidence() > .5f))

        {

            if (winkel2_degree >= 23.0 && winkel2_degree <= 50.0)
            {
                lContact = true;

                if (contact_left)
                {
                    drink = true;
                    lDrinking = true;

                }
                else
                {
                    lDrinking = false;
                }
            }
            else
            {
                lDrinking = false;
                lContact = false;
            }
        }

        else
        {
            lDrinking = false;
            lContact = false;
        }
    }

    else
    {
        lDrinking = false;
        lAngle = false;
        lContact = true;
    }
}

```

- Hier werden zuerst wieder Ober- und Unterarm als Vektoren berechnet (hier a und b, und exemplarisch für die Linke Seite),

```

cv::Point3d L_shoulder(LShoulder.getPosition().x, LShoulder.getPosition().y,
LShoulder.getPosition().z);

cv::Point3d L_elbow(LElbow.getPosition().x, LElbow.getPosition().y,
LElbow.getPosition().z);

cv::Point3d L_hand(LHand.getPosition().x, LHand.getPosition().y,
LHand.getPosition().z);

cv::Vec3d L_Ober(L_elbow.x - L_shoulder.x, L_elbow.y - L_shoulder.y, L_elbow.z
- L_shoulder.z);
cv::Vec3d L_Unter(L_hand.x - L_elbow.x, L_hand.y - L_elbow.y, L_hand.z -
L_elbow.z);

```

→ Welche normalisiert, d.h. auf die Länge 1 gebracht werden:

```
cv::Vec3d L_Oberarm = normalize(L_Ober);
cv::Vec3d L_Unterarm = normalize(L_Unter);
```

→ Dann Erfolgt die Berechnung vom Abstand der linken und rechten Hand zum Kopf:

```
cv::Vec3d head_to_lefthand(L_hand.x - head.x, L_hand.y - head.y, L_hand.z - head.z);
```

```
double distance_head_left_Hand = sqrt(pow(head_to_lefthand[0], 2) + pow(head_to_lefthand[1], 2) + pow(head_to_lefthand[2], 2));
```

→ Sowie die Berechnung des Winkels zwischen Ober und Unterarm über die Formel:

$$\alpha = \arctan(a * b)$$

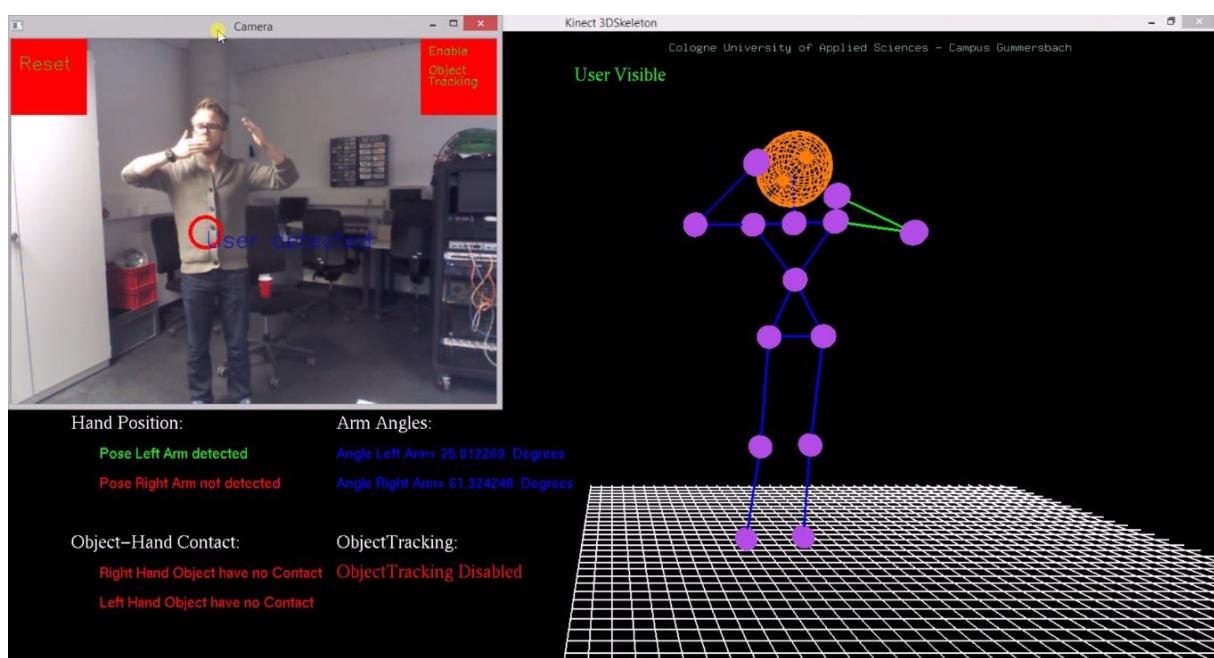
```
double winkel2_radian = acos(L_Oberarm[0] * L_Unterarm[0] + L_Oberarm[1] * L_Unterarm[1] + L_Oberarm[2] * L_Unterarm[2]);
```

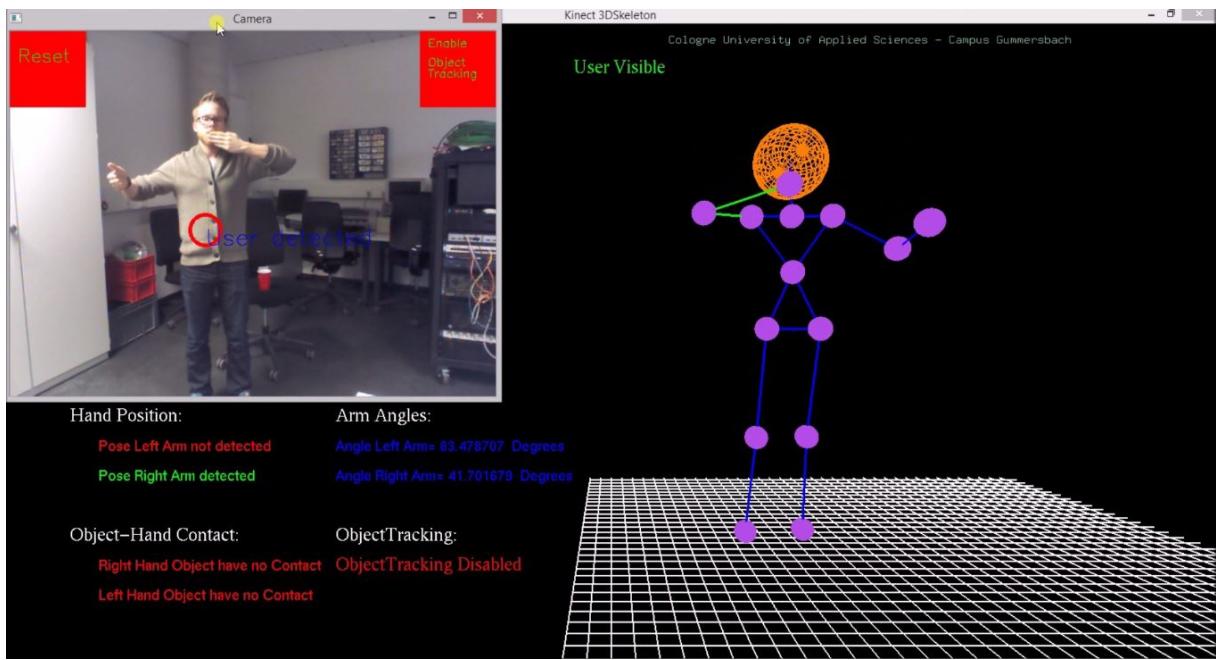
→ Diese Winkel werden nun noch von Bogen in Gradmaß umgerechnet zur besseren Veranschaulichung:

```
winkel2_degree = 180 - (180 / M_PI * winkel2_radian);
```

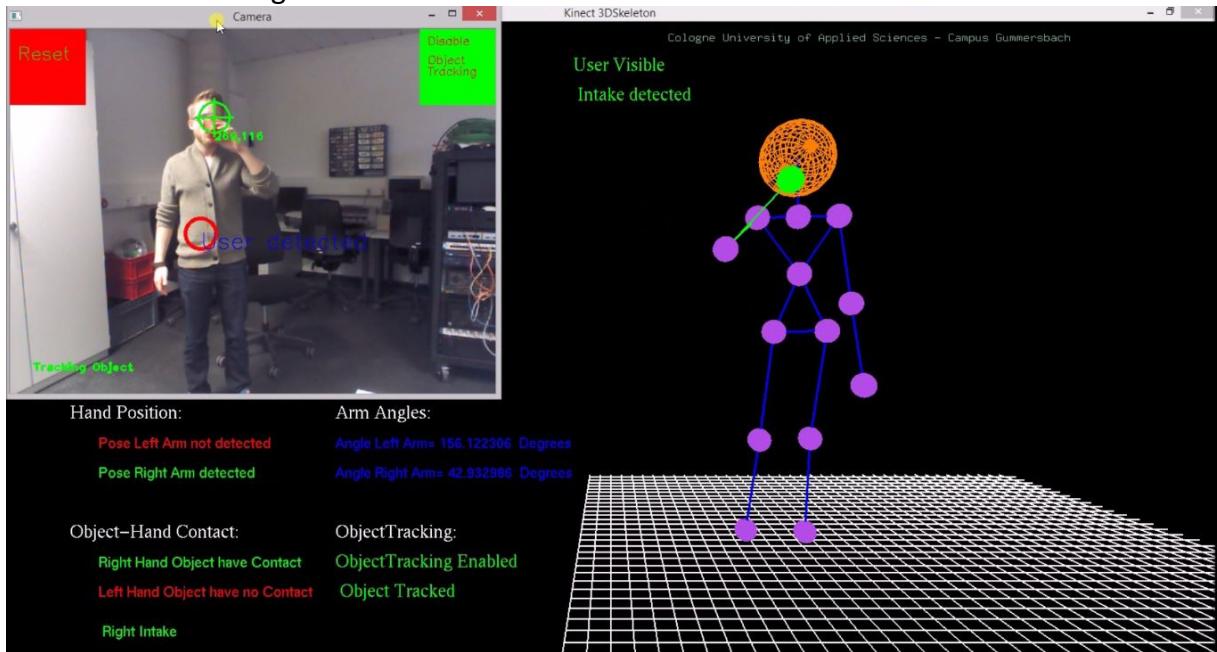
Nun erfolgt nur noch eine kleine Logik durch if- Abfragen welche den Status:

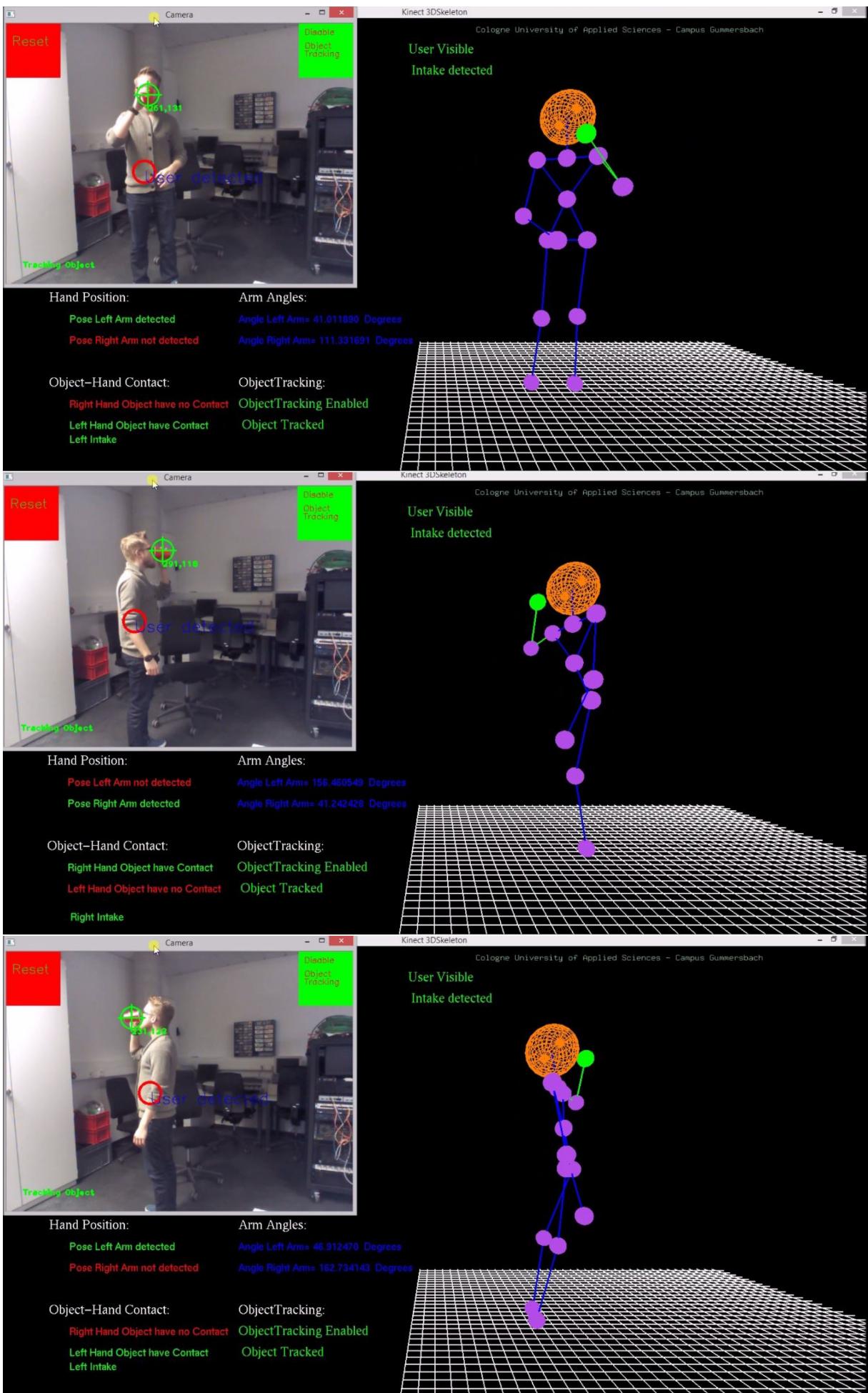
- `if (distance_head_left_Hand < 400 && Head.getPositionConfidence() > .5f && (LHand.getPositionConfidence() > .5f))`
- Wenn die Berechnung der Koordinaten mit bestimmter Wahrscheinlichkeit Erfolgt ist Und ein bestimmter Abstand von Hand und Kopf erreicht wird
- `if (winkel2_degree >= 23.0 && winkel2_degree <= 50.0)`
- Ist zudem ein Winkel in einem Bestimmten Bereich detektiert?
 - `lContact = true;` Bestätige Kontakt von Arm und Kopf.

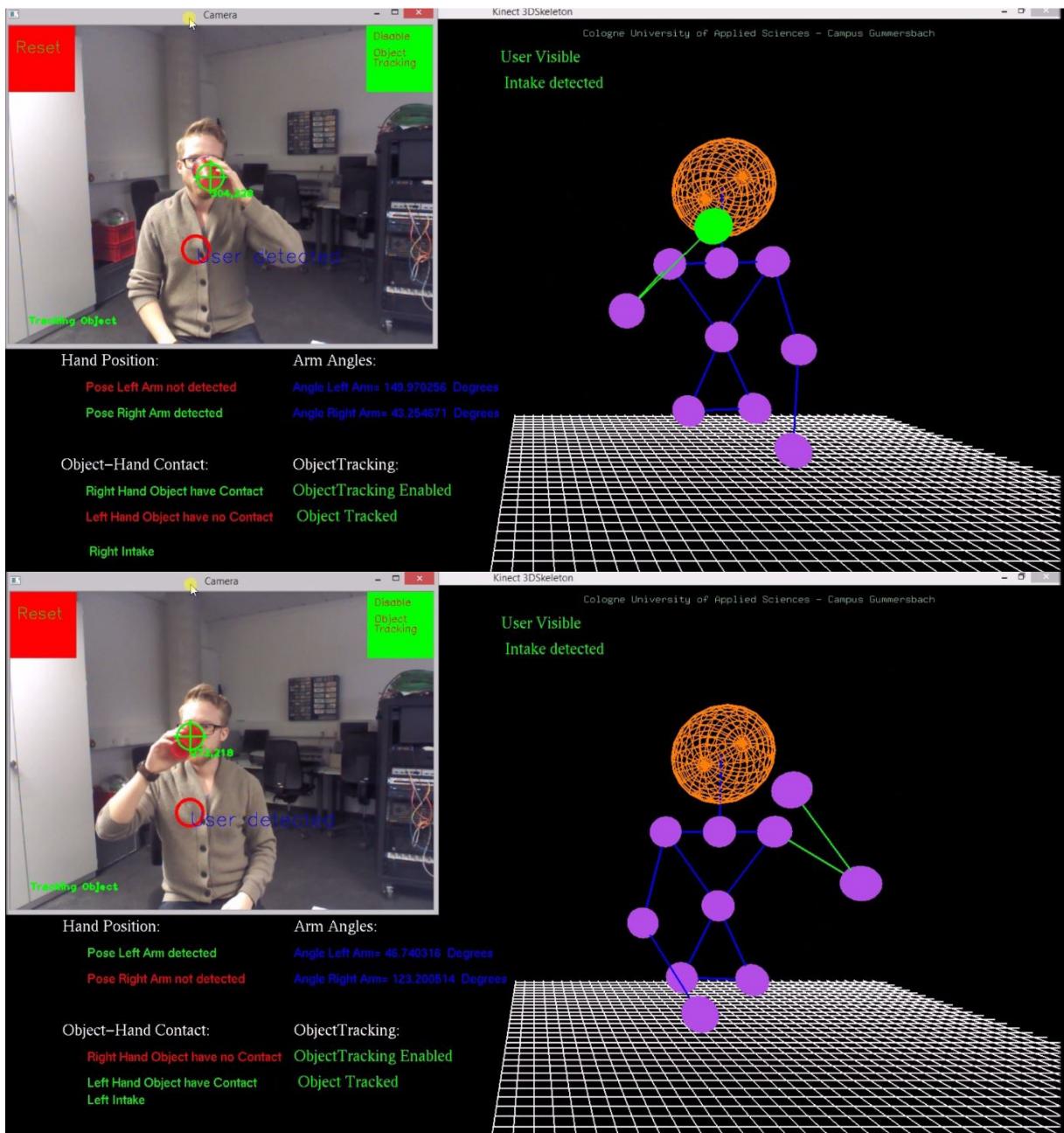




- ```
if (contact_left)
{
 drink = true;
 lDrinking = true;
```
- Sollte auch ein Kontakt der entsprechenden Hand zum Objekt detektiert sein Wird die Pose bestätigt.







Über Alle Berechnungen und Detektionen wird im Programmfenster ein Visuelles Feedback gegeben. Zudem „merkt“ sich das Programm ob eine Einnahme in der Programmlaufzeit stattgefunden hat. Dieser Wert kann durch Gesten optional zurückgesetzt werden. Genauso Auch kann auch die Objektdetektierung ein- und ausgeschaltet werden.

**Alle Berechnungen und Visualisierungen finden in Echtzeit statt!**

