**Na konci súboru je skopírovaná celá trieda Helpers.php, sú tam všetky metódy, čo vykonávajú výpočty. Útržky kódov v každom parametri nižšie používajú niektorú z metód v tej triede Helpers.php**

**Transversal výpočty nefungujú správne**

**Shoulder Abduction / Adduction Transversal**

**PHP Nuitrack:**

* Maximálne vychýlenie 180°
* Vzorec: Uhol medzi dvoma súradnícami (Shoulder a Elbow) po XZ osi. Podmienka v tomto kóde slúži na detekciu, či elbow a torso majú rovnobežnú hodnotu Z, ak áno, tak budeme od 180 odpočítavať tú hodnotu. Je to kvôli tomu, že namiesto toho ,aby uhol sa zvyšoval keď Shoulder Abduction prejde určitý bod, tak sa uhol znižoval (išiel teda 90, 80, 70.. namiesto 90, 100, 110)

if ($data[Helpers::*$JOINTS*[$side . "Elbow"]]->real->Z - $data[Helpers::*$JOINTS*["Torso"]]->real->Z >= 0) {  
 $val = 180 - abs(abs(Helpers::***getAngle2XZ***($data[Helpers::*$JOINTS*[$side . "**Shoulder**"]]->real, $data[Helpers::*$JOINTS*[$side . "**Elbow**"]]->real)) - 90);  
} else {  
 $val = abs(abs(Helpers::***getAngle2XZ***($data[Helpers::*$JOINTS*[$side . "**Shoulder**"]]->real, $data[Helpers::*$JOINTS*[$side . "**Elbow**"]]->real)) - 90);  
}

**C# Kinect:**

* Vzorec: Uhol medzi dvoma súradnícami (Shoulder a Elbow) po XZ osi. Podmienka v tomto kóde slúži na detekciu, či elbow a torso majú rovnobežnú hodnotu Z, ak áno, tak budeme od 180 odpočítavať tú hodnotu. Je to kvôli tomu, že namiesto toho ,aby uhol sa zvyšoval keď Shoulder Abduction prejde určitý bod, tak sa uhol znižoval (išiel teda 90, 80, 70.. namiesto 90, 100, 110)

var shoulderJoint = skeleton.GetJoint(side == Side.Left ? JointId.ShoulderLeft : JointId.ShoulderRight);

var elbowJoint = skeleton.GetJoint(side == Side.Left ? JointId.ElbowLeft : JointId.ElbowRight);

var torsoJoint = skeleton.GetJoint(JointId.SpineChest);

if (elbowJoint.Position.Z - torsoJoint.Position.Z >= 0)  
{  
 return 180 - Math.Abs(Math.Abs(Helpers.GetAngle2XZ(shoulderJoint.Position, elbowJoint.Position)) – 90);  
}  
else  
{  
 return Math.Abs(Math.Abs(Helpers.GetAngle2XZ(shoulderJoint.Position, elbowJoint.Position)) – 90);  
}

**Shoulder Abduction / Adduction Frontal**

**PHP Nuitrack:**

* Maximálne vychýlenie 180°
* Rotácia po osi Z

$val = (Helpers::***getZRotation***($data[Helpers::*$JOINTS*[$side . "**Shoulder**"]]->orient->Matrix) -  
 (($side == "Right") ? + 90 : -90));

**C# Kinect:**

* Maximálne vychýlenie 90°
* Ako v predchadzajúcej metóde, vychýlenie sa po dosiahnutí 90° začne zase zmenšovať
* Adduction má zápornú hodnotu
* Vzorec: Rotácia ramena po osi Z

var shoulderJoint = skeleton.GetJoint(side == Side.Left ? JointId.ShoulderLeft : JointId.ShoulderRight);

return Helpers  
 .GetZRotation(Matrix4x4.CreateFromQuaternion(shoulderJoint.Quaternion)) + 90;

**Shoulder Flexion / Extension**

**PHP Nuitrack:**

* Maximálne vychýlenie 180°
* Vzorec: hodnota rotačnej matice bodu Shoulder po **Y** osi
* Podobná situácia ako pri Shoulder Transversal

if ($data[Helpers::*$JOINTS*[$side . "Elbow"]]->real->Y - $data[Helpers::*$JOINTS*[$side . "Shoulder"]]->real->Y >= 0) {  
 $val = 180 - abs(Helpers::***getYRotation***($data[Helpers::*$JOINTS*[$side . "**Shoulder**"]]->orient->Matrix));  
} else {  
 $val = abs(Helpers::***getYRotation***($data[Helpers::*$JOINTS*[$side . "**Shoulder**"]]->orient->Matrix));  
}

**C# Kinect:**

* Vzorec: hodnota rotačnej matice bodu Shoulder po **X** osi

var shoulderJoint = skeleton.GetJoint(side == Side.Left ? JointId.ShoulderLeft : JointId.ShoulderRight);

var elbowJoint = skeleton.GetJoint(side == Side.Left ? JointId.ElbowLeft : JointId.ElbowRight);

if (elbowJoint.Position.X - shoulderJoint.Position.X >= 0)

{

return Math.Abs(Helpers.GetXRotation(  
 Matrix4x4.CreateFromQuaternion(shoulderJoint.Quaternion)));

}  
else  
{  
 return 180 – Math.Abs(Helpers.GetXRotation(  
 Matrix4x4.CreateFromQuaternion(shoulderJoint.Quaternion)));  
}

**Elbow Flexion / Extension**

**PHP Nuitrack:**

* 0° - 20° ak nie je žiadne zovretie v lakti
* Maximálne vychýlenie 180°
* Vzorec: uhol medzi troma bodmi (Elbow, Shoulder a Wrist)

$val = Helpers::***getAngle3***(  
 $data[Helpers::*$JOINTS*[$side . "**Elbow**"]]->real,  
 $data[Helpers::*$JOINTS*[$side . "**Shoulder**"]]->real,  
 $data[Helpers::*$JOINTS*[$side . "**Wrist**"]]->real  
 ) - 180;

**C# Kinect:**

* Vzorec: uhol medzi troma bodmi (Elbow, Shoulder a Wrist)

var shoulderJoint = skeleton.GetJoint(side == Side.Left ? JointId.ShoulderLeft : JointId.ShoulderRight);

var elbowJoint = skeleton.GetJoint(side == Side.Left ? JointId.ElbowLeft : JointId.ElbowRight);

var wristJoint = skeleton.GetJoint(side == Side.Left ? JointId.WristLeft : JointId.WristRight);

return 180 - Helpers.GetAngle3(elbowJoint.Position, shoulderJoint.Position, wristJoint.Position);

**Hip Flexion / Extension**

**PHP Nuitrack:**

* 0° - 20° ak človek normálne rovno stojí
* Maximálne vychýlenie 180°
* Vzorec: Uhol medzi dvomi súradnicami (YZ) bodov Knee a Waist (panva)

$val = (Helpers::***getAngle2YZ***(  
 $data[Helpers::*$JOINTS*[$side . "**Knee**"]]->real,  
 $data[Helpers::*$JOINTS*["**Waist**"]]->real  
 ) - 90);

**C# Kinect:**

* Knee extension ma zápornú hodnotu
* Vzorec: Uhol medzi dvomi súradnicami (YZ) bodov Knee a Waist (panva)

var kneeJoint = skeleton.GetJoint(side == Side.Left ? JointId.KneeLeft : JointId.KneeRight);

var waistJoint = skeleton.GetJoint(JointId.Pelvis);

return Helpers.GetAngle2YZ(kneeJoint.Position, waistJoint.Position) + 90;

**Knee Flexion / Extension**

**PHP Nuitrack:**

* 0° - 20° ak človek normálne rovno stojí
* Maximálne vychýlenie 180°
* Vzorec: Rovnaký vzorec ako pri Elbow Flexion, čiže uhol medzi 3 bodmi (Knee, Hip, Ankle)

$val = ((Helpers::***getAngle3***(  
 $data[Helpers::*$JOINTS*[$side . "**Knee**"]]->real,  
 $data[Helpers::*$JOINTS*[$side . "**Hip**"]]->real,  
 $data[Helpers::*$JOINTS*[$side . "**Ankle**"]]->real  
 ))) - 180;

**C# Kinect:**

* Vzorec: Rovnaký vzorec ako pri Elbow Flexion, čiže uhol medzi 3 bodmi (Knee, Hip, Ankle)

var kneeJoint = skeleton.GetJoint(side == Side.Left ? JointId.KneeLeft : JointId.KneeRight);

var hipJoint = skeleton.GetJoint(side == Side.Left ? JointId.HipLeft : JointId.HipRight);

var ankleJoint = skeleton.GetJoint(side == Side.Left ? JointId.AnkleLeft : JointId.AnkleRight);

return 180 - Helpers.GetAngle3(kneeJoint.Position, hipJoint.Position, ankleJoint.Position);

**Hip Abduction / Adduction Frontal**

**PHP Nuitrack:**

* 0° - 20° ak človek normálne rovno stojí
* Maximálne vychýlenie 180°
* Vzorec: hodnota rotačnej matice bodu Hip po **Z** osi

$val = ((Helpers::***getZRotation***(  
 $data[Helpers::*$JOINTS*[$side . "**Hip**"]]->orient->Matrix  
)));

**C# Kinect:**

* Abduction aj adduction su kladne hodnoty
* Vzorec: hodnota rotačnej matice bodu Hip po **Y** osi

var hipJoint = skeleton.GetJoint(side == Side.Left ? JointId.HipLeft : JointId.HipRight);

return Helpers.GetYRotation(Matrix4x4.CreateFromQuaternion(hipJoint.Quaternion)) + 90;

**Hip Abduction / Adduction Transversal**

**PHP Nuitrack:**

* 0° - 20° ak človek normálne rovno stojí
* Maximálne vychýlenie 180°
* Vzorec: hodnota medzi dvoma súradnicami Hip a Knee XZ

$val = abs(abs(Helpers::***getAngle2XZ***($data[Helpers::*$JOINTS*[$side . "**Hip**"]]->real, $data[Helpers::*$JOINTS*[$side . "**Knee**"]]->real)) - 90);

**C# Kinect:**

* Vzorec: hodnota medzi dvoma súradnicami Hip a Knee XZ

var kneeJoint = skeleton.GetJoint(side == Side.Left ? JointId.KneeLeft : JointId.KneeRight);

var hipJoint = skeleton.GetJoint(side == Side.Left ? JointId.HipLeft : JointId.HipRight);

return Math.Abs(Math.Abs(Helpers.GetAngle2XZ(hipJoint.Position, kneeJoint.Position)) - 90);

**Knee Abduction / Adduction**

**PHP Nuitrack:**

* 0° - 20° ak človek normálne rovno stojí
* Maximálne vychýlenie 180°
* Vzorec: rozdiel dvoch rotačných matíc bodov Hip a Knee po osi X

$val = ((Helpers::***getXRotationsDiff***(  
 $data[Helpers::*$JOINTS*[$side . "**Hip**"]]->orient->Matrix,  
 $data[Helpers::*$JOINTS*[$side . "**Knee**"]]->orient->Matrix  
)));

**C# Kinect:**

* Vzorec: rozdiel dvoch rotačných matíc bodov Hip a Knee po osi X

var kneeJoint = skeleton.GetJoint(side == Side.Left ? JointId.KneeLeft : JointId.KneeRight);

var hipJoint = skeleton.GetJoint(side == Side.Left ? JointId.HipLeft : JointId.HipRight);

return Helpers.GetXRotationsDiff(  
 Matrix4x4.CreateFromQuaternion(hipJoint.Quaternion), Matrix4x4.CreateFromQuaternion(kneeJoint.Quaternion));

**Spine Rotation**

**PHP Nuitrack:**

* 0° - 20° ak človek normálne rovno stojí
* Maximálne vychýlenie 180°
* Vzorec: hodnota medzi dvoma súradnicami Torso a Waist YZ

$val = (Helpers::***getAngle2YZ***(  
 $data[Helpers::*$JOINTS*["**Torso**"]]->real,  
 $data[Helpers::*$JOINTS*["**Waist**"]]->real  
 ) + 90) \* -1;

**C# Kinect:**

* rotácia napravo aj naľavo vracia kladnú hodnotu
* Vzorec: Rozdiel rotačných matíc panvy a hrude po osi Y

var waistJoint = skeleton.GetJoint(JointId.Pelvis);

var torsoJoint = skeleton.GetJoint(JointId.SpineChest);

return Helpers.GetYRotationsDiff(  
 Matrix4x4.CreateFromQuaternion(torsoJoint.Quaternion), Matrix4x4.CreateFromQuaternion(waistJoint.Quaternion));

**Thoracic Flexion**

**PHP Nuitrack:**

* 0° - 20° ak človek normálne rovno stojí
* Maximálne vychýlenie 180°
* Vzorec: Uhol medzi dvomi súradnicami (YZ) bodov neck a Torso

$val = Helpers::***getAngle2YZ***(  
 $data[Helpers::*$JOINTS*["**Neck**"]]->real,  
 $data[Helpers::*$JOINTS*["**Torso**"]]->real  
 ) + 90;

**C# Kinect:**

* Flexia má kladnú hodnotu, extenzia zápornú
* Vzorec: Uhol medzi dvomi súradnicami (YZ) bodov neck a Torso

var neckJoint = skeleton.GetJoint(JointId.Neck);

var torsoJoint = skeleton.GetJoint(JointId.SpineChest);

return (Helpers.GetAngle2YZ(neckJoint.Position, torsoJoint.Position) - 90) \* -1;

**Spine Lateroflexion**

**PHP Nuitrack:**

* 0° - 20° ak človek normálne rovno stojí
* Maximálne vychýlenie 180°
* Vzorec: hodnota rotačnej matice bodu Torso po **Z** osi

$val = Helpers::***getZRotation***(  
 $data[Helpers::*$JOINTS*["**Torso**"]]->orient->Matrix  
);

**C# Kinect:**

* Obidva pohyby majú kladnú hodnotu
* Vzorec: hodnota rotačnej matice bodu Torso po **Y** osi

var torsoJoint = skeleton.GetJoint(JointId.SpineChest);

return Helpers.GetYRotation(

Matrix4x4.CreateFromQuaternion(torsoJoint.Quaternion)) + 90;

**Cervical Flexion**

**PHP Nuitrack:**

* 0° - 2° ak človek normálne rovno stojí
* Maximálne vychýlenie o cca 30° na obidva smery
* Vzorec: Uhol medzi dvoma súradnicami (YZ) bodov Head, Neck

$val = (Helpers::***getAngle2YZ***(  
 $data[Helpers::*$JOINTS*["**Head**"]]->real,  
 $data[Helpers::*$JOINTS*["**Neck**"]]->real  
 ) + 90) \* -1;

**C# Kinect:**

* Flexia má kladnú hodnotu, extenzia zápornú
* Vzorec: Uhol medzi dvoma súradnicami (YZ) bodov Head, Neck

var neckJoint = skeleton.GetJoint(JointId.Neck);

var headJoint = skeleton.GetJoint(JointId.Head);

return (Helpers.GetAngle2YZ(headJoint.Position, neckJoint.Position) - 90) \* -1;

**Lumbar Flexion**

**PHP Nuitrack:**

* 0° - 2° ak človek normálne rovno stojí
* Vzorec: Uhol medzi dvoma súradnicami (YZ) bodov Waist, Neck

$val = (Helpers::***getAngle2YZ***(  
 $data[Helpers::*$JOINTS*["**Neck**"]]->real,  
 $data[Helpers::*$JOINTS*["**Waist**"]]->real  
 ) + 90) \* -1;

**C# Kinect:**

* Flexia má kladnú hodnotu, extenzia zápornú
* Vzorec: Uhol medzi dvoma súradnicami (YZ) bodov Waist, Neck

var neckJoint = skeleton.GetJoint(JointId.Neck);

var pelvisJoint = skeleton.GetJoint(JointId.Pelvis);

return (Helpers.GetAngle2YZ(neckJoint.Position, pelvisJoint.Position) - 90) \* -1;

**Helpers.php**

<?php  
  
namespace App\Http\Controllers\BodyMotion\Calculate;  
  
class Helpers  
{  
 public static *$JOINTS* = array(  
 "Head" => 1,  
 "Neck" => 2,  
 "Torso" => 3,  
 "Waist" => 4,  
 "LeftCollar" => 5,  
 "LeftShoulder" => 6,  
 "LeftElbow" => 7,  
 "LeftWrist" => 8,  
 "LeftHand" => 9,  
 "LeftFinertip" => 10,  
 "RightCollar" => 11,  
 "RightShoulder" => 12,  
 "RightElbow" => 13,  
 "RightWrist" => 14,  
 "RightHand" => 15,  
 "RightFingertip" => 16,  
 "LeftHip" => 17,  
 "LeftKnee" => 18,  
 "LeftAnkle" => 19,  
 "LeftFoot" => 20,  
 "RightHip" => 21,  
 "RightKnee" => 22,  
 "RightAnkle" => 23,  
 "RightFoot" => 24,  
 );  
  
 public static function getXRotation($matrix)  
 { //0 3 6  
 return round(atan2($matrix[7], $matrix[8]) \* (180 / pi()));  
 }  
  
 public static function getYRotation($matrix)  
 { // 1 4 7  
 $sy = sqrt($matrix[0] \* $matrix[0] + $matrix[3] \* $matrix[3]);  
 return round(atan2(-$matrix[6], $sy) \* (180 / pi()));  
 }  
  
 public static function getZRotation($matrix)  
 { // 2 5 8  
 return round(atan2($matrix[3], $matrix[0]) \* (180 / pi()));  
 }  
  
 public static function GetCMDifference($a, $b)  
 {  
 return round(($a - $b) / 10, 1);  
 }  
  
 public static function getAngle3($AR, $BR, $CR)  
 {  
 $A = array($AR->X, $AR->Y, $AR->Z);  
 $B = array($BR->X, $BR->Y, $BR->Z);  
 $C = array($CR->X, $CR->Y, $CR->Z);  
  
 $ab = self::*dist*($A, $B);  
 $bc = self::*dist*($B, $C);  
 $ac = self::*dist*($A, $C);  
  
 $radians = (pow($ab, 2) + pow($bc, 2) - pow($ac, 2)) /  
 (2 \* $ab \* $bc);  
  
 $targetToParentVector = self::*calculateVector*($A, $B);  
 $targetToChildVector = self::*calculateVector*($A, $C);  
  
 $dot = self::*dot*($targetToParentVector, $targetToChildVector);  
 $maga = sqrt(  
 pow($targetToParentVector[0], 2) +  
 pow($targetToParentVector[1], 2) +  
 pow($targetToParentVector[2], 2)  
 );  
 $magb = sqrt(  
 pow($targetToChildVector[0], 2) +  
 pow($targetToChildVector[1], 2) +  
 pow($targetToChildVector[2], 2)  
 );  
  
 return round(acos($dot / ($maga \* $magb)) \* (180 / pi()));  
  
 $abNorm = array();  
  
 $radians = acos(self::*dot*($targetToParentVector, $targetToChildVector) /  
 (count($targetToParentVector) \* count($targetToChildVector)));  
 return round($radians \* (180 / pi()));  
 }  
  
 private static function dist($p1, $p2)  
 {  
 return sqrt(  
 pow($p1[0] - $p2[0], 2) +  
 pow($p1[1] - $p2[1], 2) +  
 pow($p1[2] - $p2[2], 2)  
 );  
 }  
  
 private static function calculateVector($V1, $V2)  
 {  
 $res = array();  
 for ($i = 0; $i < count($V1); $i++) {  
 $res[$i] = $V1[$i] - $V2[$i];  
 }  
 return $res;  
 }  
  
 public static function getAngle3XY($P1, $P2, $P3)  
 {  
 $v = atan2($P3->Y - $P1->Y, $P3->X - $P1->X) -  
 atan2($P2->Y - $P1->Y, $P2->X - $P1->X);  
 return round($v \* (180 / pi()));  
 }  
  
 public static function getAngle3YZ($P1, $P2, $P3)  
 {  
 $v = atan2($P3->Y - $P1->Y, $P3->Z - $P1->Z) -  
 atan2($P2->Y - $P1->Y, $P2->Z - $P1->Z);  
 return round($v \* (180 / pi()));  
 }  
  
 public static function getAngle3XZ($P1, $P2, $P3)  
 {  
 $v = atan2($P3->Z - $P1->Z, $P3->X - $P1->X) -  
 atan2($P2->Z - $P1->Z, $P2->X - $P1->X);  
 return round($v \* (180 / pi()));  
 }  
  
 private static function norm($vec)  
 {  
 $norm = 0;  
 $components = count($vec);  
  
 for ($i = 0; $i < $components; $i++)  
 $norm += $vec[$i] \* $vec[$i];  
  
 return sqrt($norm);  
 }  
  
 private static function dot($vec1, $vec2)  
 {  
 $prod = 0;  
 $components = count($vec1);  
  
 for ($i = 0; $i < $components; $i++)  
 $prod += ($vec1[$i] \* $vec2[$i]);  
  
 return $prod;  
 }  
  
 public static function getXRotationsDiff($M1, $M2)  
 {  
 return self::*getXRotation*($M1) - self::*getXRotation*($M2);  
 }  
  
 public static function getYRotationsDiff($M1, $M2)  
 {  
 return self::*getYRotation*($M1) - self::*getYRotation*($M2);  
 }  
  
 public static function getZRotationsDiff($M1, $M2)  
 {  
 return self::*getZRotation*($M1) - self::*getZRotation*($M2);  
 }  
  
 public static function getAngle2XY($A, $B)  
 {  
 $xDiff = $B->X - $A->X;  
 $yDiff = $B->Y - $A->Y;  
 return round(atan2($yDiff, $xDiff) \* 180.0 / pi());  
 }  
  
 public static function getAngle2XZ($A, $B)  
 {  
 $xDiff = $B->X - $A->X;  
 $yDiff = $B->Z - $A->Z;  
 return round(atan2($yDiff, $xDiff) \* 180.0 / pi());  
 }  
  
 public static function getAngle2YZ($A, $B)  
 {  
 $xDiff = $B->Z - $A->Z;  
 $yDiff = $B->Y - $A->Y;  
 return round(atan2($yDiff, $xDiff) \* (180.0 / pi()));  
 }  
  
 public static function avg($a)  
 {  
 return round(array\_sum($a) / count($a), 1);  
 }  
  
 public static function med($arr)  
 {  
 $num = count($arr);  
 $middleVal = floor(($num - 1) / 2);  
 if ($num % 2) {  
 return $arr[$middleVal];  
 } else {  
 $lowMid = $arr[$middleVal];  
 $highMid = $arr[$middleVal + 1];  
 return round((($lowMid + $highMid) / 2));  
 }  
 }  
  
 public static function getShare($arr, $from, $to)  
 {  
 $count = 0;  
 foreach ($arr as $val) {  
 if (abs(round($val, 0)) >= $from and abs(round($val, 0)) <= $to)  
 $count = $count + 1;  
 }  
 $all = count($arr);  
 $countSeconds = round(($count \* 250) / 1000, 1);  
 if ($count > 0)  
 return round(($count \* 100) / $all, 0) . "% - " . $countSeconds . " s";  
 return "";  
 }  
  
 public static function rangeOfMotion($arr)  
 {  
 return round(abs(min($arr))) . "° - " . round(abs(max($arr))) . "°";  
 }  
  
}

**Helpers.cs**

using System;

using System.Linq;

using System.Numerics;

namespace BodyMotionData

{

internal class Helpers

{

/// <summary>

/// Positions: 0 3 6

/// </summary>

/// <param name="matrix"></param>

/// <returns></returns>

internal static double GetXRotation(Matrix4x4 matrix)

{

return Math.Round(Math.Atan2(matrix.M32, matrix.M33) \* (180 / Math.PI));

}

/// <summary>

/// Positions: 1 4 7

/// </summary>

/// <param name="matrix"></param>

/// <returns></returns>

internal static double GetYRotation(Matrix4x4 matrix)

{

var sy = Math.Sqrt(matrix.M11 \* matrix.M11 + matrix.M21 \* matrix.M21);

return Math.Round(Math.Atan2(-matrix.M31, sy) \* (180 / Math.PI));

}

/// <summary>

/// Positions: 2 5 8

/// </summary>

/// <param name="matrix"></param>

/// <returns></returns>

internal static double GetZRotation(Matrix4x4 matrix)

{

return Math.Round(Math.Atan2(matrix.M21, matrix.M11) \* (180 / Math.PI));

}

internal static double GetCMDifference(double a, double b)

{

return Math.Round((a - b) / 10, 1);

}

// TODO: this function is not written correctly in php

internal static double GetAngle3(Vector3 v1, Vector3 v2, Vector3 v3)

{

var a = new Vector3(v1.X, v1.Y, v1.Z);

var b = new Vector3(v2.X, v2.Y, v2.Z);

var c = new Vector3(v3.X, v3.Y, v3.Z);

var targetToParentVector = CalculateVector(a, b);

var targetToChildVector = CalculateVector(a, c);

var dot = Dot(targetToParentVector, targetToChildVector);

var maga = Math.Sqrt(

Math.Pow(targetToParentVector.X, 2)

+ Math.Pow(targetToParentVector.Y, 2)

+ Math.Pow(targetToParentVector.Z, 2)

);

var magb = Math.Sqrt(

Math.Pow(targetToChildVector.X, 2)

+ Math.Pow(targetToChildVector.Y, 2)

+ Math.Pow(targetToChildVector.Z, 2)

);

return Math.Round(Math.Acos(dot / (maga \* magb)) \* (180 / Math.PI));

}

internal static double GetDistance(Vector3 v1, Vector3 v2)

{

return Math.Sqrt(

Math.Pow(v1.X - v2.X, 2)

+ Math.Pow(v1.Y - v2.Y, 2)

+ Math.Pow(v1.Z - v2.Z, 2)

);

}

internal static Vector3 CalculateVector(Vector3 v1, Vector3 v2)

{

return new Vector3(

v1.X - v2.X,

v1.Y - v2.Y,

v1.Z - v2.Z

);

}

internal static double GetAngle3XY(Vector3 v1, Vector3 v2, Vector3 v3)

{

var v = Math.Atan2(v3.Y - v1.Y, v3.X - v1.X) - Math.Atan2(v2.Y - v1.Y, v2.X - v1.X);

return Math.Round(v \* (180 / Math.PI));

}

internal static double GetAngle3YZ(Vector3 v1, Vector3 v2, Vector3 v3)

{

var v = Math.Atan2(v3.Y - v1.Y, v3.Z - v1.Z) - Math.Atan2(v2.Y - v1.Y, v2.Z - v1.Z);

return Math.Round(v \* (180 / Math.PI));

}

internal static double GetAngle3XZ(Vector3 v1, Vector3 v2, Vector3 v3)

{

var v = Math.Atan2(v3.Z - v1.Z, v3.X - v1.X) - Math.Atan2(v2.Z - v1.Z, v2.X - v1.X);

return Math.Round(v \* (180 / Math.PI));

}

internal static double Norm(Vector3 v)

{

return Math.Sqrt(Vector3.Dot(v, v));

}

internal static double Dot(Vector3 v1, Vector3 v2)

{

return Vector3.Dot(v1, v2);

}

internal static double GetXRotationsDiff(Matrix4x4 matrix1, Matrix4x4 matrix2)

{

return GetXRotation(matrix1) - GetXRotation(matrix2);

}

internal static double GetYRotationsDiff(Matrix4x4 matrix1, Matrix4x4 matrix2)

{

return GetYRotation(matrix1) - GetYRotation(matrix2);

}

internal static double GetZRotationsDiff(Matrix4x4 matrix1, Matrix4x4 matrix2)

{

return GetZRotation(matrix1) - GetZRotation(matrix2);

}

internal static double GetAngle2XY(Vector3 a, Vector3 b)

{

var xDiff = b.X - a.X;

var yDiff = b.Y - a.Y;

return Math.Round(Math.Atan2(yDiff, xDiff) \* 180.0 / Math.PI);

}

internal static double GetAngle2XZ(Vector3 a, Vector3 b)

{

var xDiff = b.X - a.X;

var zDiff = b.Z - a.Z;

return Math.Round(Math.Atan2(zDiff, xDiff) \* 180.0 / Math.PI);

}

internal static double GetAngle2YZ(Vector3 a, Vector3 b)

{

var zDiff = b.Z - a.Z;

var yDiff = b.Y - a.Y;

return Math.Round(Math.Atan2(yDiff, zDiff) \* 180.0 / Math.PI);

}

internal static double Avg(double[] a)

{

return Math.Round(a.Sum() / a.Length, 1);

}

// TODO: this function is not written correctly in php

internal static double Med(double[] arr)

{

double num = arr.Length;

var middleVal = (int)Math.Round(num / 2);

if (num % 2 != 0)

{

return arr[middleVal];

}

else

{

var lowMid = arr[middleVal];

var highMid = arr[middleVal + 1];

return Math.Round((lowMid + highMid) / 2);

}

}

internal static string GetShare(double[] arr, int from, int to)

{

double count = 0;

foreach (var val in arr)

{

if (Math.Abs(Math.Round(val, 0)) >= from && Math.Abs(Math.Round(val, 0)) <= to)

{

count++;

}

}

var all = arr.Length;

var countSeconds = Math.Round(count \* 250 / 1000, 1);

if (count > 0)

{

return Math.Round(count \* 100 / all, 0) + "% - " + countSeconds + "s";

}

return "";

}

internal static string RangeOfMotion(double[] arr)

{

return Math.Round(Math.Abs(arr.Min())) + "° - " + Math.Round(Math.Abs(arr.Max())) + "°";

}

/// <summary>

/// Get speed of a joint in m/s between two positions

/// </summary>

/// <param name="position1"></param>

/// <param name="position2"></param>

/// <param name="time">Time in ms</param>

/// <returns></returns>

internal static double GetJointSpeed(Vector3 position1, Vector3 position2, double time)

{

var changeX = position1.X - position2.X;

var changeY = position1.Y - position2.Y;

var changeZ = position1.Z - position2.Z;

var velocityX = changeX / time;

var velocityY = changeY / time;

var velocityZ = changeZ / time;

return Math.Sqrt(velocityX \* velocityX + velocityY \* velocityY + velocityZ \* velocityZ);

}

}

}