PRZEGLĄD TELEINFORMATYCZNY
NR 1-2, 2016

Human motion capture using Kinect and third party sensors

Witold ŻORSKI, Tomasz PAŁYS

Instytut Teleinformatyki i Automatyki WAT,
ul. Gen. S. Kaliskiego 2, 00-908 Warszawa
witold.zorski@wat.edu.pl , tomasz.palys@wat.edu.pl

ABSTRACT: The paper describes an applicable solution combining advantages of the Kinect device and properties of appropriable third party sensors. The elaborated solution allows tracking of a human arm along with recognition of basic hand gestures. This way it may be possible to remotely control a manipulator or a robotic arm performing some actions determined by a user's hand. The Kinect is mainly used to preliminary calibrate the system and for verification purposes. The system was designed using kinematics-based approach with rigid transformation combining rotations and translations. Matrix transformation operators were exchanged by dual quaternions as quaternions are native for the used devices. Additionally, as this is not a trivial mathematical tool, the machinery of dual quaternions has been introduced and its implementation is given.

KEYWORDS: Kinect, space sensors, robot kinematics, rigid transformation, dual quaternions

1. Introduction

The concept of a device allowing to recognize body shape with a camera technology inspired Microsoft to create Kinect (2010). Kinect can be considered as a system that can track the movement of objects and individuals in three dimensions, interpret specific gestures, making completely hands-free control and interaction with a console or a computer [20]. The impact of such a device was immediate and overwhelming – it attracted researchers around the world, especially in robotics [4] and computer vision [8] fields, and programming communities that have released various SDK for the Kinect [9] covering essential issues: preprocessing, object tracking and recognition, human activity analysis, gesture analysis, and others. Soon after the launch of the Kinect lots of effort have been devoted to applications of this device; e.g.: indoor 3-D mapping

(by Philipp Robbel of MIT), or to control a browser with hand gestures using JavaScript extension (http://depthjs.media.mit.edu/).

A scientific adventure with the Kinect should begin from papers that may serve as tutorials: [17] – while building a new Kinect-based system, [18] considers calibration of the Kinect imaging sensor, [11] as a source of references for Kinect-based computer vision researchers, or [10] for a comparison of gesture recognition techniques in term of accuracy and efficiency.

The area of robotics is very popular in some Kinect purposes, for instance, robot motion control methods in real-time using the Kinect-based hand tracking [25], intuitive for an operator gesture based telemanipulation of an industrial robotic arm requiring minimal amount of training [22], navigation of mobile robots [5], object detection and recognition [24], scene reconstruction [26], 3D inspection [16] and others [23]. Some of these problems are convergent with the subject of this paper.

The effort of searching new application for the Kinect continues unabated. For the last year alone, we can find valuable publications on new, sometimes a little surprising frontiers: Kinect depth sensors use for object detection for vehicle collision avoidance [1], an evaluation of effectiveness of Kinect and Kinect 2 for recognition of specialized karate techniques [7], or a Kinect-based real-time interactive control system design to control a robotic arm [2], that is similar to the issue presented in this paper.

In this paper we want to propose and describe a system based on the Kinect device assisted by appropriable third party sensors, that would be responsible for enhancing movement tracking of a human arm as well as recognition of basic hand gestures, using new sensor "Myo armband", with the efficiency suitable to remotely control a manipulator or a robotic arm equipped with a tool, what is the main improvement with reference to [19]. The idea of the Kinect device supported by additional sensors looks like a negation of the Kinect nature, but the device fails for some "configurations" of a human posture (generally, it is required to be in front of the cameras), environment objects are serious obstacles (if a body part is covered), precision[1] is limited (Kinect depth measurements is proportional to the distance squared), and its space range is relatively narrow (a few meters); these seem to be inevitable problems.

The essential innovation (with reference to [19]) in the mathematical approach is the application of dual quaternions and not standard matrix transformation operators, in the case of kinematics formulas. This approach is supported by the fact that sensors return rotational quaternions that can be easily adopted in the case of dual quaternions (see Eq. 5). As dual quaternions are not trivial and being a very important aspect the machinery of the tool is explained

[1] http://wiki.ros.org/openni_kinect/kinect_accuracy

along with its easy-to-understand Scilab implementation in background.

Fig. 1 presents a man wearing two sensors of the first kind (on the waistline, and on the right arm treated as a manipulator link) and one sensor of the second kind (on the forearm). The left arm's configuration is mapped by the system along with recognized hand gestures interpreted as basic actions to be performed by a robotic manipulator. The intermediate stage is the graphical model presented in the considered figure.

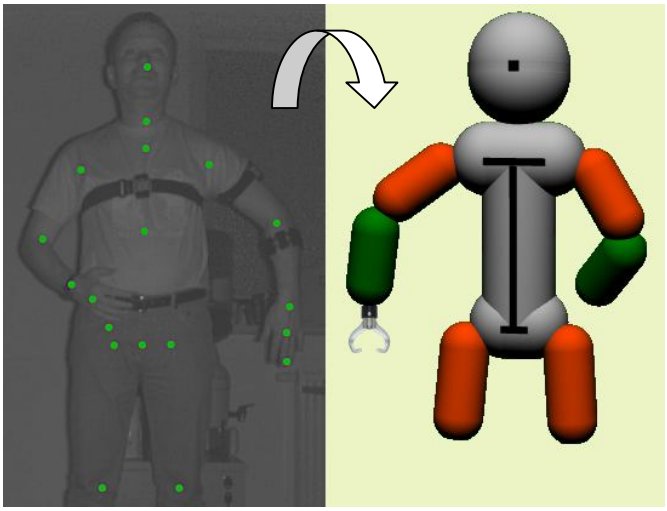

Fig. 1. Tracking and mapping an arm including hand gestures recognition

2. The computer vision system

2.1. A General View of the System

Considering the hardware the proposed computer vision system consists of a standard x86 computer equipped with the Kinect for Xbox One device, which is supported by two kinds of third party sensors – the first is the YEI 3-Space sensors; the second is the Myo armband. As for the software side the system is built with the use of Microsoft Visual Studio 2015 supported by .NET Framework 4.6.1. Fig. 2 presents a visual conception or a general view of the proposed computer vision system and its components, suggesting a possible application.

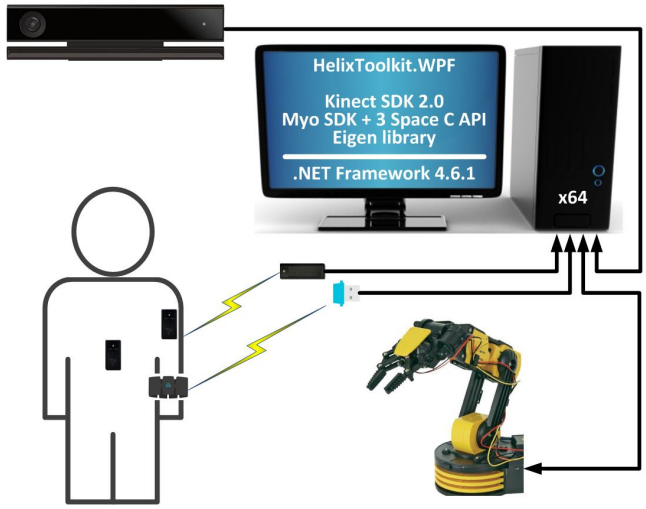

Fig. 2. A concept of the proposed system of tracking and mapping an arm

2.2. The Kinect

The proposed system uses Kinect for Xbox One – an updated version of Kinect for Windows (being well described in [5]). The new Kinect has a wide-angle time-of-flight camera, greater accuracy with 3-times the fidelity over its predecessor, and can track without visible light by using an active IR sensor. It has a wider field of vision that can detect a user up to 3 feet from the sensor, and can track up to 6 skeletons at once. It can also detect player's facial expression, the position and orientation of 25 individual joints, speed of movements, and track gestures performed with a standard controller.

Unfortunately, formal technical data (with respect to accuracy) of the device are not available. The experimental data received by ROS.org (Robot Operating System) confirm that the expected error of depth measurements is proportional to the distance squared (~ dist^2 *0.005 [m]).

The assumption is that the system will be able to perform tracking of a human arm and recognition of some basic hand gestures with the use of Kinect and additional sensors. The Kinect will be used at the stage of calibration and later for verification purposes. To perform the calibration of the system a person with sensors have to conduct a simple procedure (described in subchapter 5.1). Starting from that point the Kinect will be used for verification purposes only.

2.3. Third party sensors

To support the Kinect two kinds of third party sensors were selected within the wide market of wearable sensors: the YEI 3-Space sensors and the Myo armband.

The YEI 3-Space Sensor (shown in Fig. 3) is a miniature AHRS/IMU[2], high-precision, high-reliability single unit. This device uses gyroscope, accelerometer, and compass sensors in conjunction with advanced processing and on-board quaternion-based algorithms to determine relative to an absolute reference or a designated reference orientation in real-time. The gradient descent calibration process and high update rates increase the accuracy and reduce and compensate errors. The YEI 3-Space Sensor unit features are accessible via an open communication protocol that allows access to all sensor data and configuration parameters. Versatile commands allow access to raw and normalized sensor data, filtered absolute and relative orientation outputs given in quaternions and Euler angles (i.e. pitch/roll/yaw).

Basic parameters of the YEI 3-Space Sensor (important for the considered system) are as follows: orientation range 360º about all axes, accuracy ±1º for dynamic conditions and all orientations, repeatability 0.085º for all orientations.

Fig. 3. YEI 3-Space sensor (features: 2 input buttons and RGB indicator)

The Myo armband is a combination of wearable gesture control and motion control features in a single device (see Fig. 4). The band contains a highly-sensitive nine-axis IMU containing three-axis gyroscope, three-axis accelerometer, three-axis magnetometer.

The Myo parameters related to orientation accuracy are not available, but a simple experiment with these two devices banded together indicated that their orientation accuracies are very similar (difficult to distinguish).

[2] Attitude and Heading Reference System / Inertial Measurement Unit

Fig. 4. Myo armband device

The Myo armband tracks the electrical signal in an user's arm and detects gestures. There are five hand gestures (see Fig. 5) that can be relatively easy detected by Myo: making a fist, spreading fingers, waving left, waving right, and tapping twice thumb and index finger.

Fig. 5. Basic pre-set gestures for the Myo armband

3. Software components

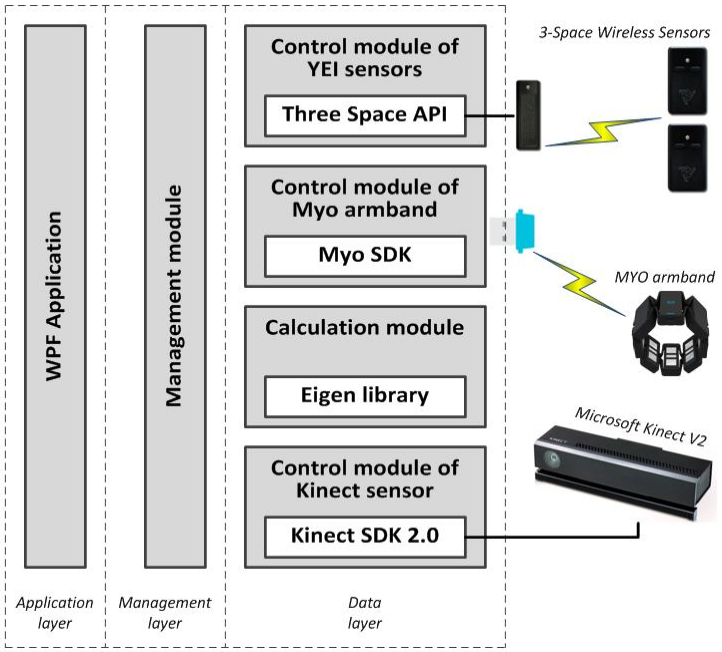

Fig. 6. The structure of the system software environment

Considering Visual Studio 2015 as the primal software environment the following three essential layers of the system can be distinguished: "application layer", "management layer", and "data layer". The structure of the software environment is presented in Fig. 6.

The application layer is equivalent to the created WPF application [21], responsible for every activity aspect of the system, particularly for the GUI. The management layer ensures management system tools that are included in a management module library. The data-layer is a set of libraries that includes: control module of the Kinect sensor (Microsoft Kinect for Windows SDK 2.0), calculation module (Eigen library 3.2.8), control module of Myo armband (Myo SDK 0.9.0), control module of YEI sensors (3-Space C API ver. 2.0.6).

The use of **Microsoft Visual Studio 2015** as a development tool was involved by the need to utilize the Kinect for Windows SDK and assets of the Windows Presentation Foundation. The WPF offers a consistent programming model for developing applications with a graphical subsystem. An additional very important aspect was the possibility of integration Microsoft .NET Common Language Runtime (a managed code) with the 3-Space C API library and the Eigen library (an unmanaged code).

The Kinect for Windows SDK 2.0 enables to create applications under Visual Studio 2015 (.NET Framwork 4.6 is required) that support advanced gesture recognition using Kinect sensor technology on computers with modern Windows.

The 3-Space API was inevitable due to the use of YEI sensors. This API is available for C/C++ languages and can be used to build independent software for YEI sensors. In the case of Visual Studio environment the C API is the appropriate choice.

The Myo SDK for Windows contains binary executables (the DLLs required for the SDK on Windows), drivers for the Bluetooth dongle bundled with Myo, C/C++ header files, link-time libraries (the LIB files required to link with the SDK), and sample programs. At the core of the Myo SDK is library "libmyo" that allows applications to interact with the Myo armband device. All functionality in libmyo is exposed through a plain C API.

The Eigen library is a high-level C++ open source template library for linear algebra, vector and matrix operations, numerical solvers, and related algorithms. This clearly facilitates to achieve high performance of required calculations. In particular, the Eigen includes some very useful functions and classes for image processing.

The Helix 3D Toolkit for .NET is a collection of custom controls and helper classes. It contains controls to manipulate the camera, classes to import and export models; it provides a higher level API for working with 3D in the WPF.

4. Quaternions and dual quaternions

The quaternions – a number system that extends the complex numbers, discovered by Hamilton in 1843. Quaternions are well described in books [15] and free e-sources[3].

A quaternion is a linear combinations of the basis elements: 1, i, j, and k; what can be written as:

$$q = a + bi + cj + dk\,, \tag{1}$$

A dual quaternion is an ordered pair of quaternions being constructed from eight real parameters. It consists of two quaternions: q_r, q_d, that are called the real part and the dual part:

$$Q = q_r + q_d\varepsilon\,, \tag{2}$$

[3] http://www.e-booksdirectory.com/listing.php?category=535

Dual quaternions formulate and solve some problems more concisely, rapidly and in fewer steps, with result clearer to others, and practice with fewer lines of code effortlessly debugged (see 8. Appendix). Rigid transformations, in particular, can be represented with eight scalar variables, and combined through concatenation.

As the rigid transformation was mentioned, it seems to be appropriate to present an adequate matrix as a reference point for dual quaternions:

$$T = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma & t_x \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma & t_y \\ -s\beta & c\beta s\gamma & c\beta c\gamma & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

The presented rigid transformation matrix (where $c\alpha$ is shorthand for $\cos\alpha$ and $s\alpha$ for $\sin\alpha$, etc.) takes into account rotations by Euler angles (α,β,γ) and translations by vector $[t_x, t_y, t_z]$; if needed, reflections may be included. In order to create a matrix described by Eq. 3 the function RTM() presented in Fig. 14 may be applied.

Now, let us consider the same functionality in the case of quaternions. The dual quaternion can represent rotation by angle φ about vector v in the following way (the dual part is set to zero):

$$Q_R = [\cos(\frac{\varphi}{2}) + (\hat{v}_x i + \hat{v}_y j + \hat{v}_z k)\sin(\frac{\varphi}{2})][0,0,0,0], \tag{4}$$

If we possess Euler angles (α,β,γ), there is an easy way to receive the quaternion responsible for the rotation (performed in the order: about $0x$ by γ, about $0y$ by β, and about $0z$ by α) as follows:

$$\begin{cases} q_x = \cos(\gamma/2) + i\sin(\gamma/2) \\ q_y = \cos(\beta/2) + j\sin(\beta/2) \\ q_z = \cos(\alpha/2) + k\sin(\alpha/2) \end{cases}, \quad Q_R = [q_z \cdot q_y \cdot q_x][0,0,0,0]. \tag{5}$$

In order to represent translation by vector $[t_x, t_y, t_z]$, the real part is set to identity and the dual part should be constructed in the following way:

$$Q_T = [1,0,0,0][0, \frac{t_x}{2}, \frac{t_y}{2}, \frac{t_z}{2}]. \tag{6}$$

Finally, the rotational and translational quaternions can be combined into a single unit allowing compact representation of a rotation followed by a translation:

$$Q = Q_T \times Q_R . \tag{7}$$

At this stage one may find it helpful to take a look on function DualQuaternion() in Fig. 13, that is responsible for creating a dual quaternion according to Eq. 5-7.

The following equation defines how to transform point P into point P', using the received dual-quaternion Q:

$$P' = Q \cdot P \cdot Q^* , \tag{8}$$

where Q^* represents the conjugate of Q, both for Q considered as a quaternion and as a dual number[4]. The transformation described by Eq. 8 has been implemented as function DQ_transformation() presented in Fig. 13 along with other necessary subfunctions, especially DQ_conj().

One can ask a question, why a new mathematical tool should be used? Dual quaternions can be just as efficient (if not more efficient) as matrix methods. Additionally, dual quaternions are singularity-free, un-ambiguous, and give shortest path of interpolation. The tool becomes very popular in the computer world, particularly in computer graphics [12]; quaternions are almost inevitable in robotics [6][5], even the used sensors are quaternion-based devices.

5. The kinematics model

In this section a kinematics model, based on dual quaternions, for the considered system is presented. The obvious mathematical instrument appropriate to perform required transformations on data received from sensors affixed to a human stature derives from robotics, where the kinematics [14] plays the fundamental role. The used notation for the presented kinematics model is based on [3].

5.1. Initial configuration

To perform the system calibration procedure a person wearing sensors have to take a posture in front of the Kinect (the arm must be aligned to ensure proper calibration), shown in Fig. 7, and press the button on sensor S_1, what resets all sensors. This way the orientation of sensors is treated as the reference.

[4] This is the main gap (a mistake) made in the implementation for dual quaternions presented in [13].
[5] A very good paper on dual quaternions.

The very important issue is stable fixing of sensors, that will guarantee their persistent position against parts of arm (its bones); the place of fixing is not important as the sensors are reset (during calibration).

Next, the system reads coordinates of points $P_1 = [p_1^x, p_1^y, p_1^z]$ to $P_4 = [p_4^x, p_4^y, p_4^z]$ from the Kinect and the following crucial distances for a user body are calculated (see Fig. 8):

$$\begin{cases} x_{21} = p_2^x - p_1^x \\ y_{21} = p_2^y - p_1^y \end{cases}, \quad x_{32} = p_3^x - p_2^x, \quad x_{43} = p_4^x - p_3^x. \tag{9}$$

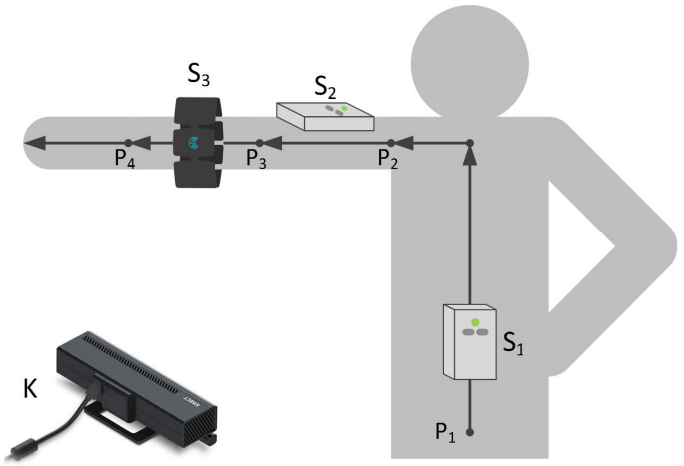

Fig. 7. The posture required during the system calibration (K – Kinect, S_1,S_2,S_3 – sensors)

5.2. Transformations

To describe the location of links (i.e. parts of a user's body) relative to its neighbors a coordinate system (a frame[6]) has been affixed to each link, as is shown in Fig. 8.

As was suggested, we are going to interchange rigid transformation matrices (a fundamental tool in book [3]) with more reliable dual quaternions. This can be performed with minimal effort as the nature of kinematics equations will stay intact (see [19] for comparison).

[6] A frame is a coordinate system with a position vector which locates its origin relative to some other embedding frame.

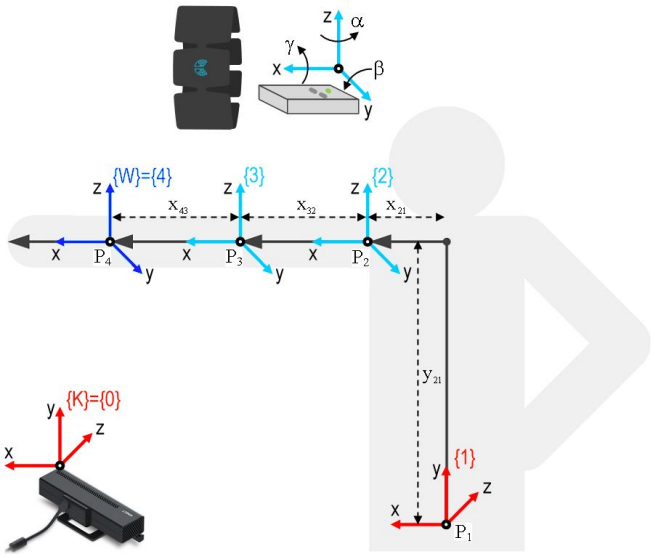

Fig. 8. A human skeleton modeled as a kinematic chain (see Fig. 7)

In such an approach, each point, particularly those received from the Kinect at the stage of calibration should be considered as a quaternion. E.g. point P_1 (see Fig. 8) may be presented as a quaternion or a dual quaternion in the following ways:

$$^{K}p_1 = [0, p_1^x, p_1^y, p_1^z],$$

$$^{K}P_1 = [1,0,0,0][0, p_1^x, p_1^y, p_1^z]. \tag{10}$$

Now, we are ready to create transformation quaternions, that are adequate to conditions presented in Fig. 8, based on Euler angles (α, β, γ) received from sensors. We will use differences between corresponding angles in the following way (an example): $\alpha_3 = \alpha_{S3} - \alpha_{S2}$.

In order to satisfy kinematic chain $\{K\} \leftrightarrow \{W\}$ the set of following dual quaternions is required:

$$^{0}_{1}Q = {}^{K}_{1}Q = [1,0,0,0][0, \frac{p_1^x}{2}, \frac{p_1^y}{2}, \frac{p_1^z}{2}], \tag{11}$$

$$^{1}_{2}Q = Q(0,0,\frac{-\pi}{2}, x_{21}, y_{21}, 0) \cdot Q(\alpha_2, \beta_2, \gamma_2, 0,0,0), \tag{12}$$

$$^{2}_{3}Q = Q(\alpha_3, \beta_3, \gamma_3, x_{32}, 0,0), \tag{13}$$

where $Q(\alpha, \beta, \gamma, t_x, t_y, t_z)$ stands for a dual quaternion generated according to Eq. 5-7 (see also function DualQuaternion() in Fig. 13). Equations 11-13 are equivalent to equations 3-5 used in [19].

In the considered system, we do not use a "palm sensor", thus the origin of frame {W} (i.e. point P_4) is the last point in the kinematic chain that can be calculated (see Eq. 16).

The obtained dual quaternions (link transformations) can be multiplied together to find the single quaternion transformation that relates frame {3} to frame {K}:

$$ {}^{K}_{3}Q = {}^{0}_{3}Q = {}^{0}_{1}Q \cdot {}^{1}_{2}Q \cdot {}^{2}_{3}Q \tag{14} $$

Analogously to Eq. 14, we can obtain any quaternion transformation ${}^{m}_{n}Q$ that relates frame {n} to frame {m} in a given chain of frames.

Finally, we can calculate coordinates of points P_2, P_3, P_4 with respect to frame {K} in the following way:

$$ \begin{cases} {}^{K}P_2 = {}^{0}_{2}Q \cdot \{[1,0,0,0][0,0,0,0]\} \cdot {}^{0}_{2}Q^{*} \\ {}^{K}P_3 = {}^{0}_{3}Q \cdot \{[1,0,0,0][0,0,0,0]\} \cdot {}^{0}_{3}Q^{*} \\ {}^{K}P_4 = {}^{0}_{3}Q \cdot \{[1,0,0,0][0,x_{43},0,0]\} \cdot {}^{0}_{3}Q^{*} \end{cases} \tag{15} $$

In a destination robotics system the "wrist" coordinates of point P_4 with respect to frame {1} will be preferred, as this is the case when the arm is fully independent of the Kinect as the calibration is done, what can be calculated from:

$$ {}^{1}P_4 = {}^{1}_{3}Q \cdot \{[1,0,0,0][0,x_{43},0,0]\} \cdot {}^{1}_{3}Q^{*} \tag{16} $$

6. Results

In order to initially check the system a series of experiment was conducted. At the beginning the graphical model (see Fig. 1) was tested independently for every part of the arm, and this way we checked the sensors as well as the proposed kinematics model link by link. Next, the graphical model was compared with coordinates of points received from the Kinect. Finally, we performed some experiments critical for the Kinect, when a participant was back towards the Kinect, moving out of its range, or partly covered by something.

The results we obtained are very similar to those presented previously in [19], and that was expected. A sample of conducted verification tests is presented in Fig. 9–11, respectively for coordinates X, Y, and Z. The considered

figures present wrist coordinates values (in metres) received during a person activity along an example period of 15+10 seconds. The period of 10 extra seconds was performed in such a way the user body was moving out of the Kinect range. This is the reason why coordinates received from the Kinect are strongly concurrent with those calculated by the system only along 15 seconds.

An important issue is the result of hand gestures recognition performed by the Myo armband. In that case we find out[7] it doesn't recognize those gestures correctly 100% of the time – it got gestures right closer to range of **70-80%** of the time.

The system was checked in practice using robot Mentor. We conducted a simple experiment – its arm was considered as a two-link planar manipulator with an impactive end-effector which physically grasp. The configuration of the arm was mapped on the robot several times per second (by reading coordinates of a human wrist and solving the inverse kinematics problem). In order to perform grasp action we used only "fist" gesture for the Myo armband as it was best recognizable by the device (above 80%). In such a simple scenario the mapping process proved to be successful if there is a possibility of correcting the process on-the-fly by robot observation.

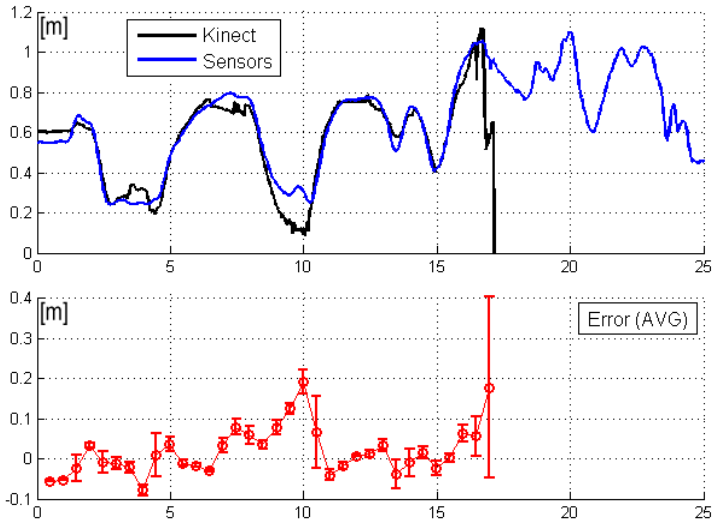

Fig. 9. Wrist coordinate X – values and the average error

[7] Based on a procedure carried out several times by two participants.

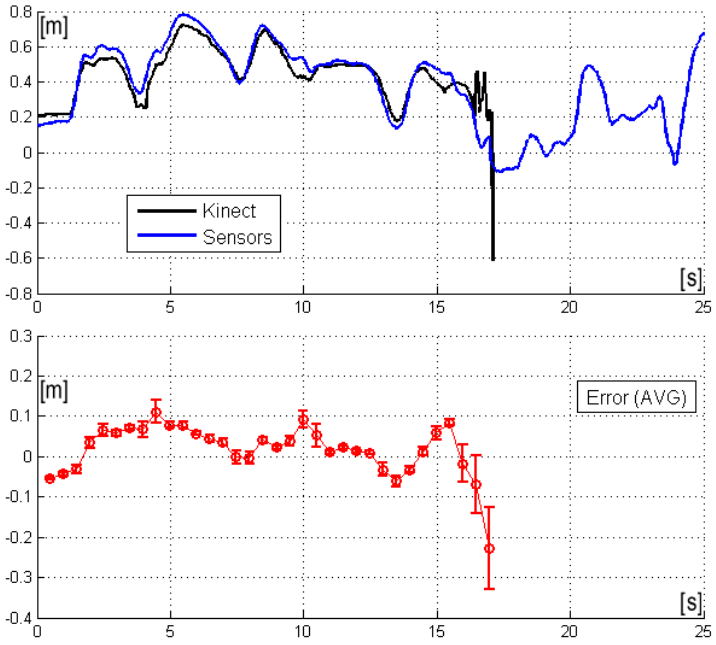

Fig. 10. Wrist coordinate Y – values and the average error

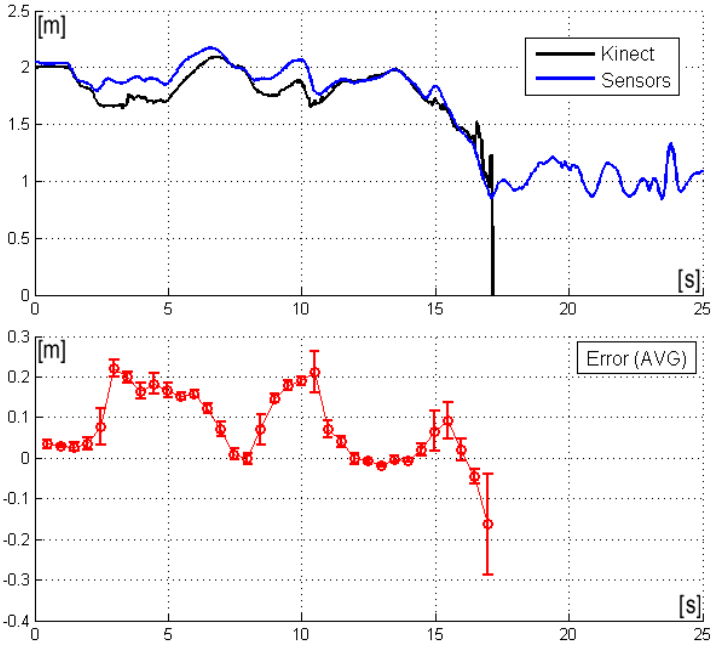

Fig. 11. Wrist coordinate Z – values and the average error

7. Conclusion

The idea of using third party sensors along with the Kinect device turned out to be justified as some of Kinect adversities (the device fails for some "configurations" of a human posture – see Fig. 9-11, some objects can be serious obstacles, operating space is narrowed) has been reduced.

There are important gains of the work that should be mentioned. Thanks to high-precision sensors the system has a high level of accuracy (limited mainly by calibration) along with a stable wide operating range (limited by sensors' WiFi) and high immunity to terrain obstacles. As sensors are concerned, the Myo armband proved to be a promising forerunner of the future wearable technologies design for human-computer interaction. The problem with Myo is that it is limited to a few gestures, that sometimes can actually be uncomfortable to perform (e.g. a hand perpendicular to arm is far from being natural).

We used only quaternions, even in the case of data received from the sensors as they are native for these devices. The authors' Scilab implementation (see appendix) of dual quaternions was carefully tested, verified on theoretical models, and can be easily adopted in another system or converted to any programming language if needed (that is proved by our system with the software written in C).

In order to perform advanced robotics experiments the system can be improved in terms of accuracy by the use of modern and smaller sensors that will remove some technical limitations, and further simplified by revamping the calibration process. To perform somewhat more real a robot with at least 6 degrees of freedom is required.

8. Appendix

This appendix offers an authors' implementation[8] of basic functions for quaternions (Fig. 12) and "far from being trivial" dual quaternions (Fig. 13). A simple test of correctness is also included (Fig. 15) as the dual quaternions should work the same way as the rigid transformation matrices (Fig. 14). The implementation is given in Scilab and this way is short, complete, and clear as matrices are basic objects defined in Scilab environment. It was used by authors at the initial stage and later for results verification on theoretical models.

```
//quaternions - basic functions

function [pq]=q_mult(p,q) // quaternions multiplication, p*q
    p1=p(1,1); p2=p(1,2); p3=p(1,3); p4=p(1,4);
    q1=q(1,1); q2=q(1,2); q3=q(1,3); q4=q(1,4);
    pq(1,1)=p1*q1-p2*q2-p3*q3-p4*q4;
    pq(1,2)=p2*q1+p1*q2-p4*q3+p3*q4;
    pq(1,3)=p3*q1+p4*q2+p1*q3-p2*q4;
    pq(1,4)=p4*q1-p3*q2+p2*q3+p1*q4;
endfunction

function [q_c]=q_conj(q) //conjugation
    q_c=-q; q_c(1,1)=q(1,1)
endfunction

function [q_n]=q_normalize(q) //normalization
    q_n=q/sqrt(q*q') //division by modulus
endfunction

function [q_r]=q_rotate(q,p)
    q_n=q_normalize(q);
    q_r=q_mult(q_n,q_mult(p,q_conj(q_n)));
endfunction

function [q_r]=q_rotate_RPY(a,b,g,p) //a,b,g - roll, pitch, yaw; p=[0,x,y,z]
    rz(1,1)=cos(a/2); rz(1,2)=0;        rz(1,3)=0;        rz(1,4)=sin(a/2);
    ry(1,1)=cos(b/2); ry(1,2)=0;        ry(1,3)=sin(b/2); ry(1,4)=0;
    rx(1,1)=cos(g/2); rx(1,2)=sin(g/2); rx(1,3)=0;        rx(1,4)=0;
    q_r=q_rotate(rz,q_rotate(ry,q_rotate(rx,p))); //order - rotation by: g,b,a
endfunction
```

Fig. 12. Basic functions for quaternions (in Scilab)

[8] Link to the source: https://drive.google.com/open?id=0B2teHRt0NtcYLVlRUmxtZnRkLVk

```
// dual quaternions - basic functions

function [Q]=DualQuaternion(a,b,g,tx,ty,tz) //creation
//INPUTs: a,b,g - Euler angles; [tz,ty,tx] - translation vector
    //rotation quaternion (order: 0x by g, 0y by b, 0z by a)
    qz(1,1)=cos(a/2); qz(1,2)=0;        qz(1,3)=0;        qz(1,4)=sin(a/2);
    qy(1,1)=cos(b/2); qy(1,2)=0;        qy(1,3)=sin(b/2); qy(1,4)=0;
    qx(1,1)=cos(g/2); qx(1,2)=sin(g/2); qx(1,3)=0;        qx(1,4)=0;
    Qr=g_normalize(g_mult(qz,g_mult(qy,qx)));
    //translation quaternion for vector [tx ty tz]
    Qt=[0 tx ty tz]/2;
    Qd=g_mult(Qt,Qr); //translation component (Qt multiplied by Qr)
    Q=[Qr; Qd] //real & dual parts of the created dual quaternion
endfunction

function [Q_c]=Q_conj(Q) //dual quaternion conjugation
    Q_c=[g_conj(Q(1,:));
        -g_conj(Q(2,:))] //dual conjugation -> the minus sign
endfunction

function [Q_n]=Q_normalize(Q) //dual quaternion normalization
    Q_n=Q/sqrt(Q(1,:)*Q(1,:)') //division by modulus of the real part
endfunction

function [PQ]=Q_mult(P,Q) //dual quaternions multiplication, P*Q
    PQ=[g_mult(P(1,:),Q(1,:));
        g_mult(P(1,:),Q(2,:))+g_mult(P(2,:),Q(1,:))]
endfunction

function [dq_trans]=Q_transformation(a,b,g,tx,ty,tz,p)
//a,b,g - Euler angles; [tz,ty,tx] - translation vector; point p=[0,x,y,z]
    dq=DualQuaternion(a,b,g,tx,ty,tz);
    dp=[1 0 0 0; p] //p without division by 2
    dq_trans=Q_mult(dq,Q_mult(dp,Q_conj(dq)));
endfunction
```

Fig. 13. Basic functions for dual quaternions (in Scilab)

```
function [T]=RTM(a,b,g,tx,ty,tz)  //RTM matrix creation
//INPUTs: a,b,g - Euler angles; [tz,ty,tx] - translation vector
    ca=cos(a); sa=sin(a); cb=cos(b); sb=sin(b); cg=cos(g); sg=sin(g);
    T=[ca*cb  ca*sb*sg-sa*cg  ca*sb*cg+sa*sg  tx;
       sa*cb  sa*sb*sg+ca*cg  sa*sb*cg-ca*sg  ty;
       -sb    cb*sg           cb*cg           tz;
       0      0               0               1]
endfunction
```

Fig. 14. Rigid transformation matrix (in Scilab)

```
//Euler angles (a - roll, b - pitch, g - yaw)
disp('Euler angles:')
a=30*%pi/180; disp(a*180/%pi,'a=')
b=45*%pi/180; disp(b*180/%pi,'b=')
g=60*%pi/180; disp(g*180/%pi,'g=')

//translation
tx=10; ty=20; tz=30;
disp([tz,ty,tx],'translation vector [tx,ty,tz]=')

//input vector (a point)
P0=[3; 4; 5; 1] //x,y,z,1 - for RTM
disp(P0,'vector P0=')
p0=[0 P0(1,1) P0(2,1) P0(3,1)] //0,x,y,z - for quaternions
disp(p0,'point P0 as quaternion p0=')

disp('RESULTs')
disp(RTM(a,b,g,tx,ty,tz)*P0,'for matrices: T*P0=')
Q=Q_normalize(Q_transformation(a,b,g,tx,ty,tz,p0));
Qt=q_mult(Q(2,:),q_conj(Q(1,:))) //extraction translation
disp(Qt,'for quaternions: q*p*inv(q)=')
```

Fig. 15. A simple test of implemented functions correctness (in Scilab)

References

[1] ABDULLAH M. S. H., ZABIDI A., YASSIN I. M., HASSAN H. A., *Analysis of Microsoft Kinect depth perception for distance detection of vehicles*. IEEE 6th Control and System Graduate Research Colloquium, ICSGRC 2015, pp. 116-119.

[2] BENABDALLAH I., BOUTERAA Y., BOUCETTA R., REKIK C., *Kinect-based Computed Torque Control for lynxmotion robotic arm*. 7th International Conference on Modelling, Identification and Control, ICMIC 2015, pp. 1-6.

[3] CRAIG J. J., *Introduction to Robotics: mechanics and control*. Addison-Wesley Publishing Company, Singapore, 1989.

[4] EL-LAITHY R.A., HUANG J., YEH M., *Study on the use of Microsoft Kinect for robotics applications*. Position Location and Navigation Symposium (PLANS), 2012, pp. 1280-1288.

[5] FANKHAUSER P., et al., *Kinect v2 for Mobile Robot Navigation: Evaluation and Modeling*. IEEE International Conference on Advanced Robotics, ICAR 2015, pp. 388-394.

[6] GOUASMI M., OUALI M., BRAHIM F., *Robot Kinematics Using Dual Quaternions*. International Journal of Robotics and Automation (IJRA), Vol. 1, No. 1, March 2012, pp. 13-30.

[7] HACHAJ T., OGIELA M. R., KOPTYRA K., *Effectiveness Comparison of Kinect and Kinect 2 for Recognition of Oyama Karate Techniques*. 18th International Conference on Network-Based Information Systems, NBiS, 2015, pp. 332-337.

[8] HAN J., SHAO L., XU D., SHOTTON J., *Enhanced Computer Vision With Microsoft Kinect Sensor: A Review*. IEEE Transactions on Cybernetics, Vol. 43, No. 5., 2013, pp. 1318-1334.

[9] ISLAM M. R., et al., *A Novel Approach for Constructing Emulator for Microsoft Kinect XBOX 360 Sensor in the .NET Platform*. 4th International Conference on Intelligent Systems, Modeling and Simulation, 2013, pp. 1-6.

[10] JAIS H. M., MAHAYUDDIN Z. R., ARSHAD H., *A review on gesture recognition using Kinect*, International Conference on Electrical Engineering and Informatics, ICEEI 2015, pp. 594-599.

[11] JUNGONG H., LING S, DONG XU, SHOTTON J., *Enhanced Computer Vision with Microsoft Kinect Sensor: A Review*. IEEE Transactions on Cybernetics, Vol. 43(5), 2013, pp. 1318-1334.

[12] KAVAN L., COLLINS S., ZARA J., O'SULLIVAN C., *Skinning with dual quaternions*. Proceedings of the 2007 symposium on Interactive 3D graphics and games, I3D 2007, pp. 39-46.

[13] KENWRIGHT B., *A Beginners Guide to Dual-Quaternions*. WSCG 2012 Communication Proceedings, pp.1-13.

[14] KIM JUNG-HA, KUMAR V. R., *Kinematics of robot manipulators via line transformations*. Journal of Robotic Systems, Vol. 7(4), 1990, pp. 649–674.

[15] KUIPERS J. B., *Quaternions and Rotation Sequences: A Primer with Applications to Orbits*. Aerospace and Virtual Reality, Princeton University Press, 2002.

[16] MAJDI A., BAKKAY M. C., ZAGROUBA E., *3D modeling of indoor environments using Kinect sensor*. Second International Conference on Image Information Processing, ICIIP 2013, pp. 67-72.

[17] MING A., ENOMOTO K., SHINOZAKI M., SATO R., SHIMOJO M., *Development of an entertainment robot system using Kinect*. 8th Europe-Asia Congress on Mechatronics, 2014, pp. 127-132.

[18] PAGLIARI D., MENNA F., RONCELLA R., REMONDINO F., PINTO L., *Kinect Fusion improvement using depth camera calibration*. International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XL-5, 2014, pp. 479-485.

[19] PAŁYS T., ŻORSKI W., *Enhanced movement tracking with Kinect supported by high-precision sensors*. Proceedings of the Federated Conference on Computer Science and Information Systems 2015, pp. 883-888.

[20] PATSADU O., NUKOOLKIT C. WATANAPA B., *Human gesture recognition using Kinect camera*. The Proceeding of International Joint Conference on Computer Science and Software Engineering, JCSSE 2012, pp. 28-32.

[21] REN YU, GUBING LU, FENG LU, *A Method of Rotation Transformation for 3D Object by Changing Camera Attributes in WPF*. International Conference on Information Engineering and Computer, 2010, pp. 1-4.

[22] SHIRWALKAR S., SINGH A., SHARMA K., SINGH N., *Telemanipulation of an industrial robotic arm using gesture recognition with Kinect*. International Conference on Control, Automation, Robotics and Embedded Systems, CARE 2013, pp. 1-6.

[23] STONE E. E., et al., *Evaluation of the Microsoft Kinect for screening ACL injury*. 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBC 2013, pp. 4152-4155.

[24] WANG T., BU L., HUANG Z., *A new method for obstacle detection based on Kinect depth image*. Chinese Automation Congress (CAC), 2015, pp. 537-541.

[25] WU H., et al., *Kinect-based robotic manipulation: From human hand to end-effector*. IEEE 10th Conference on Industrial Electronics and Applications, ICIEA 2015, pp. 806-811.

[26] WU L., CHAI S., *Depth Filtering in 3D Reconstruction of Indoor Scenes Based on Kinect*. Seventh International Symposium on Computational Intelligence and Design, ISCID 2014, pp. 356-359.

Przechwytywanie ruchów i gestów za pomocą Kinecta i specjalizowanych sensorów

STRESZCZENIE: Artykuł przedstawia rozwiązanie problemu śledzenia ręki poruszającego się człowieka, wraz z rozpoznawaniem prostych gestów, przy wykorzystaniu właściwości urządzenia Kinect oraz dodatkowych sensorów. Rozwiązanie może być stosowane do zdalnego sterowania manipulatorem za pomocą ręki i gestów dłoni. Urządzenie Kinect służy głównie na etapie kalibracji systemu oraz w celu weryfikacji jego działania. System został pierwotnie zaprojektowany przy użyciu standardowej kinematyki manipulatora, opartej na macierzowych operatorach przekształcenia, które następnie zostały zastąpione przez kwaterniony dualne, gdyż są one wykorzystywane natywnie przez zastosowane urządzenia. Artykuł zawiera krótkie wprowadzenie do kwaternionów dualnych oraz ich przykładową implementację w środowisku Scilab.

SŁOWA KLUCZOWE: Kinect, sensory, kinematyka manipulatora, operator przekształcenia, kwaterniony dualne.

Praca wpłynęła do redakcji: 15.09.2016 r.