A Motion Compensation Method with an UR3-Robot in Medical Applications

Nasser Attar, Mirko Schimkat, Konstantin Stepanow and Arvind Ramasamy

***Abstract*—This paper presents a method, where a robot is used to precisely compensate the body movements of a human patient during medical treatments. The method uses visual tracking to determine the position and orientation of the relevant body parts of the patient and afterwards, based on the received tracking data, the robot is moving a specific medical device, such that the device attains a commanded spatial relation to the relevant body parts of the patient. In the first section of this paper, it is shown, why such a method is useful and which fields of application exists. Furthermore, a specific medical scenario is presented, which will be used in this paper to demonstrate the motion compensation method. The second section presents the used robot and his kinematic model. The calibration procedure is treated in the third section. The fourth section introduces the visual tracking system and the calibration and tracking algorithms to estimate the posture of the patient's body. The fifth section considers the constraints and conditions, the motion of the robot is subject to and derives appropriate path planning algorithms to fulfill them. The sixth section shows how the actual motion compensation is assembled from the previous contents and the last section presents the results of an experiment, where the motion compensation had been tested.**

I. INTRODUCTION

In a number of medical applications, it is necessary to place a medical device next to a human patient in such way that the device keeps exactly a predetermined position and orientation relative to the body of the patient. One example is the placement of a radiation coil in space, next to the head of a patient, such that it can radiates a specific part of the patient's head. Naturally, a human patient is doing very small movements of his body parts, unconsciously, and therefore the considered device must follow this movements to maintain the commanded relation. Obviously, a robotic operator is suitable for executing this task. A tracking system measures the movements and posture of the patient and it does the same for the medical device. The tracking system sends this data to a PC, which processes the data, to determine which movements the end-effector of the robot must execute to maintain the specified spatial relation between the medical device and the patient. Then the PC sends movement commands to the robot and the robot is reacting with a movement of his manipulator, which carries the medical device. Of course, care must be taken that the robot or the device is not colliding with the patient. To demonstrate the above explanations the scenario shown in Fig. 1 is considered. The robot and the tracking system are shown which have their own coordinate systems attached to them. In this scenario, the only important body part of the patient is his head. The coil, which is carried by the end-effector of the robot, shall keep a predetermined posture relative to the head. To express this posture, the head and the coil also have corresponding coordinate systems attached to them. Then the relative posture between them can be expressed with the homogeneous transformation matrix ${}^{\mathrm{C}}T_{\mathrm{H}}$. Analogously, the *desired* posture between head and coil has the corresponding homogeneous transformation matrix ${}^{\mathrm{C}}T_{\mathrm{H,g}}$. The goal of the motion compensation method is to move the robot, such that the relation

$$ {}^{\mathrm{C}}T_{\mathrm{H}} = {}^{\mathrm{C}}T_{\mathrm{H,g}} \tag{1} $$

always holds. Of course, due to measurement errors of the tracking system and the finite precision of the robot, ${}^{\mathrm{H}}T_{\mathrm{C}}$ can only approximate ${}^{\mathrm{C}}T_{\mathrm{H,g}}$. The coordinate frames of the patient's head, tracking system, robot and coil have in each case a posture relative to the world coordinate frame, denoted by the axes triple $(x_{\mathrm{W}}, y_{\mathrm{W}}, z_{\mathrm{W}})$. The tracking system measures the relative postures of the head relative to the coordinate system of the tracking system $(x_{\mathrm{S}}, y_{\mathrm{S}}, z_{\mathrm{S}})$, in terms of the homogeneous transformation matrices ${}^{\mathrm{S}}T_{\mathrm{H}}$. Note, that there are two kinds of tracking systems. The first tracking system is a Microsoft KINECT camera. It only outputs images with depth information to the PC. Based on the data of the KINECT, it is necessary to estimate ${}^{\mathrm{S}}T_{\mathrm{H}}$. This is done in the motion compensation method itself. The second available tracking system is an Atracsys FusionTrack tracking camera. Its output is compiled to ${}^{\mathrm{S}}T_{\mathrm{H}}$ by an independent subroutine running on

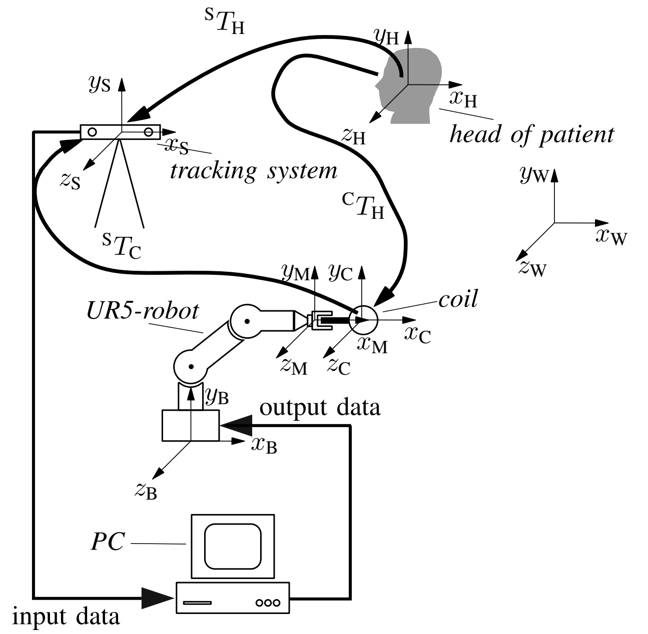


Fig. 1. Medical scenario, where the motion compensation method is used.

the PC, which means that the motion compensation algorithm has directly access to ${}^{\mathrm{S}}T_{\mathrm{H}}$. This homogeneous transformation matrix is processed together with the posture of the tracking system and of the robot in the world coordinate system to output data for the robot. This data processing is done in MATLAB. The robot will then, based on the output data from the PC, move its end-effector, such that Eq. (1) is approximated as good as possible.

II. Forward and Inverse kinematics of the UR5

Universal Robots (UR1,-3,-5,-10) are multifunctional, flexible robots, that can be used for various tasks. The number stands for the working load and, therefore, also the size of the robot. These robots are place saving and can be easily reprogrammed for new requirements [1]. The URs consist out of six rotational joints and seven fixed links. Therefore, there are eight different constellations of joint angles to reach a determined end-effector position. That means that for given joint parameters we can calculate one unique position, but for a given position there are eight different sets of angles. These eight constellations can be described as lefty/righty, up/down and flip/noflip. With lefty/righty we can distinguish between the second joint being on the right or left side of the imaginary plane going through the origin of the coordinate system of the first joint. This distinction makes sense, if the coordinate system of the second joints has an offset, according to the coordinate system of the first joint. Up/down describes the relation between the coordinate systems of the third and the fourth joint. If the fourth joint remains in the same position, the third joint can be below or above it. Therefore, we distinguish between up and down. Finally, flip/noflip describes the wrist (5th joint) being flipped or not, meaning rotated by ϕ or ϕ + 180°.

A. Forward kinematics

The forward kinematics of this kind of robot can be calculated easily with the Denavit-Hartenberg-procedure, if the parameters are known. Denavit-Hartenberg-parameters describe the relation between two coordinate systems with a maximum of four different parameters: two rotation angles and two linear translations. These can be fixed or variable. In our case, with six joints, we have seven coordinate systems and six transfomations. Coordinate system zero is not attached to any of the joints and is used as the reference frame.

B. Inverse kinematics

The inverse kinematics has been realised with the algorithm described in [4]. Its starting point is the desired pose, that we want to move our robot to. With this pose and the given Denavit-Hartenberg-parameters we calculate intermediate poses from which the six angles can be found step by step. Previously, calculated angles can furthermore be used to

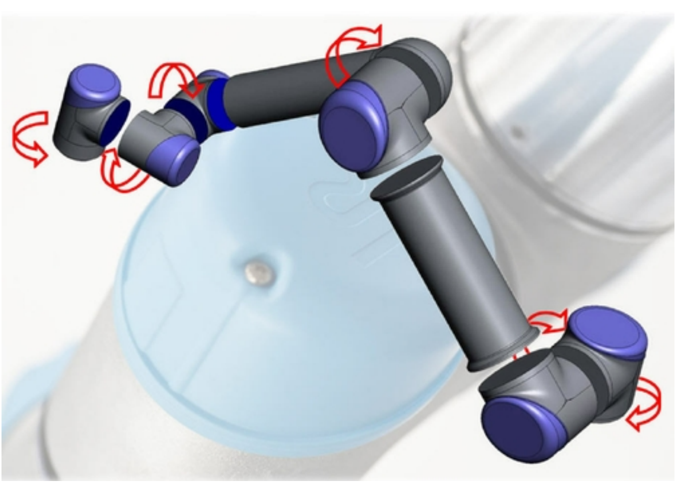


Fig. 2. Joints of the UR [2]

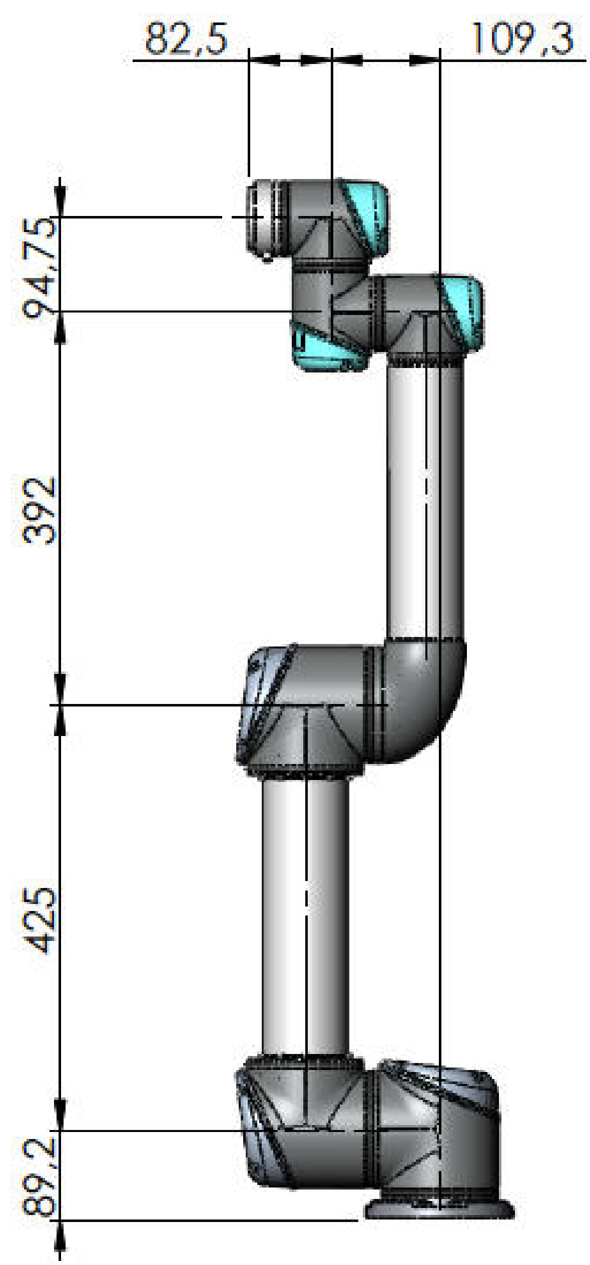


Fig. 3. Dimensions of the UR [3]

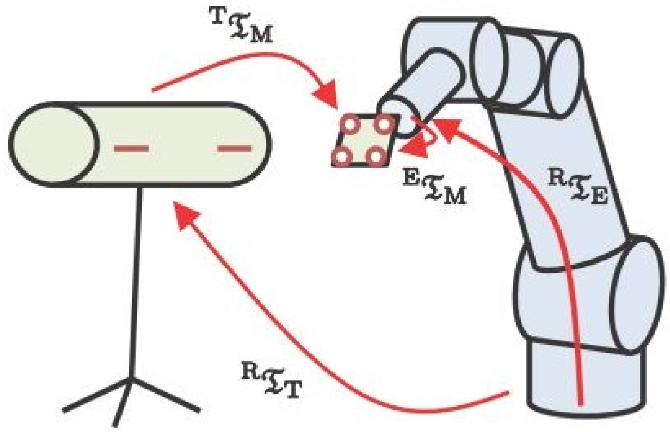


Fig. 4. Hand-Eye-calibration setup

calculate the next ones. Since there are different angles possible to reach the same pose, it is important to name the current angle that is being used to calculate the next one. Lefty/righty, up/down and flip/noflip are current names. In case that one angle has two solutions, the current solution exist with its particular name so that the distinction between different possibilities is made.

The implemented algorithm works as follows:

- Calculate eight max possible angle configurations that lead to the desired pose
- Check if the pose is possible with the robots dimensions and/or the given maximal joint changes
- Calculate the $\pm$ 360°-versions of the found angles since the robot might be able to move in the range of $\pm$ 360° and get eight new possibilities
- Choose one angle set of the 16 sets with the minimal angle sum with respect to the initial angles

III. Handy-Eye-calibration

The hand-eye-calibration is a current procedure to determine unknown rigid transformations, e.g. transformations between the end-effector and a tool attached to it or a transformation between the robot and the tracking system, by taking measurements with the tracking device in different poses of the robot. For good precision and accuracy you need many different poses and they need to be linearly independent from each other. In this project the transformation between the robot's end-effector and the coil had to be determined as well as the transformation between the robots base and the coordinate system of the tracking device. The coil needed to be placed in a certain posture with respect to the head of the imaginary patient, therefore it was not enough to know only the end-effector coordinates, that can be calculated by the forward kinematics. The hand-eye-calibration uses the closed loop of four rigid transformations. That means that we can express one transformation or a combination of two, by means of the remaining ones. Two of these transformations are fixed and the other two can vary. Fig. 4 shows the installation.

$$M =^R T_E \tag{2}$$

$$X =^E T_M \tag{3}$$

$$N =^T T_M \tag{4}$$

$$Y =^R T_T \tag{5}$$

We are interested in the fixed ones ${}^{E}\mathrm{T}_M$ and ${}^{R}\mathrm{T}_T$ whereas the variable ones can be calculated or measured. We can write

$$M \cdot X = Y \cdot N \tag{6}$$

or

$$M_i \cdot X = N_i \cdot Y \tag{7}$$

where index i denotes one set of measurements.

These equations can be rearranged and written as

$$A \cdot w = b \tag{8}$$

where A and b contain manipulated values of M_i and N_i, so that we can fill these matrices with actual measurement values to calculate w. After the calculation the solution vector w, it contains the 24 not trivial values of X and Y. The last step of the calibration is to orthonormalize X and Y since the calculation of w provides independant values for the rotation and translation. If the translation can be taken as given, the vectors of the rotation matrix need to be orthonormal [5].

IV. Tracking

The tracking of objects with the KINECT System is done in five different steps.

1) Take the Infrared (IR) image and the depth image with the KINECT System
2) Track the fiducials in the IR image
3) Convert the position of fiducials, which is given in pixels to mm
4) Identify the fiducials with respect to the reference model of the tracked object
5) Create the transformation matrix

The KINECT System takes two different images. One infrared image and one image which represents the depth. While the values of the pixels in the infrared images are a decimal value between zero and one, the values of the depth images holds the actual depth in mm. In the end, we want to estimate the position of an object. This is done by placing fiducials on the object. This fiducials reflect the infrared light in a strong way, which appears in the infrared images as a bright dot. These dots can be tracked in the images and the position can be calculated. By knowing the distances between the fiducials on the object (stored in a reference model) and comparing these distances with the measured distances between the tracked points, we can determine the actual position of the object in the three dimensional space.

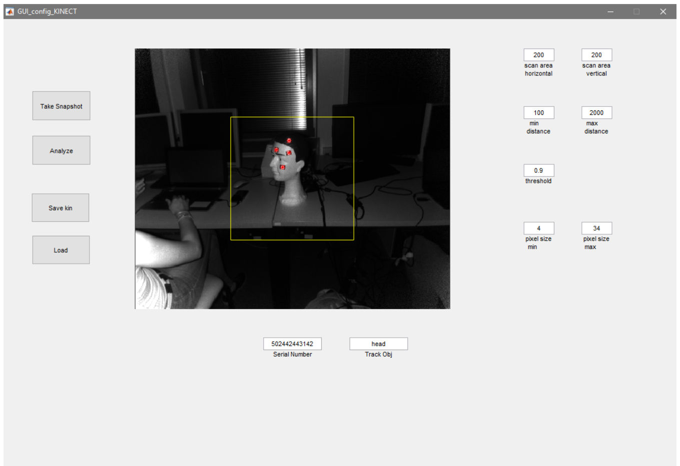


Fig. 5. The GUI with the preview of the KINECT System and the parameters.

But as in any real system, we have to deal with distortions. There are two big main problems:

1) Not only the fiducials can appear as bright dots. Also light from the sun or reflections of the infrared light from other objects can appear as bright dots in the image.
2) The distance measurement, which is used for creating the depth image, sometimes has problems to measure the right distance. In this cases, the distance for that point is zero. This is mostly the point, when a material reflect the infrared light in a strong way, which is nearly always true for the fiducials.

To track only the fiducials and not some distortions, some filtering is done. To simplify the filtering, some parameters are set before the measurement starts:

- threshold for converting IR image from grayscale to black and white
- minimal distance between object and camera in mm
- maximal distance between object and camera in millimeters
- minimal size of the fiducials in pixel
- maximal size of the fiducials in pixel
- scan area, in which the object will appear

These settings can be set via a GUI, which also shows the scan area (in yellow) and the found fiducials. This is illustrated in Fig. 5. With these parameters, the following filtering steps are performed:

1) Converting the IR image from gray scale to black and white. This is done by interpreting every pixel which is below the brightness the threshold as black and every pixel with a brightness over or equal the threshold as white.
2) Every white pixel which is to near to the camera or to wide from the camera is deleted (switched to black). For the actual distance, the depth image is taken.
3) Every white pixel outside of the scan area is turned to black.

These three steps are shown in Fig. 6, where step 1 is the original IR image, step 2 is the image converted to black and white, step 3 is the image, where the pixels which are to near or to wide away are deleted (note, that because of distortions in the depth image, the pixels in the lower left of the image are still there) and in step 4 all pixels outside of the can area are deleted. In the next processing step, the pixels are clustered. The clustering groups all pixel, which are connected to each other to one group (see Fig. 7a). For each cluster, the following values are calculated:

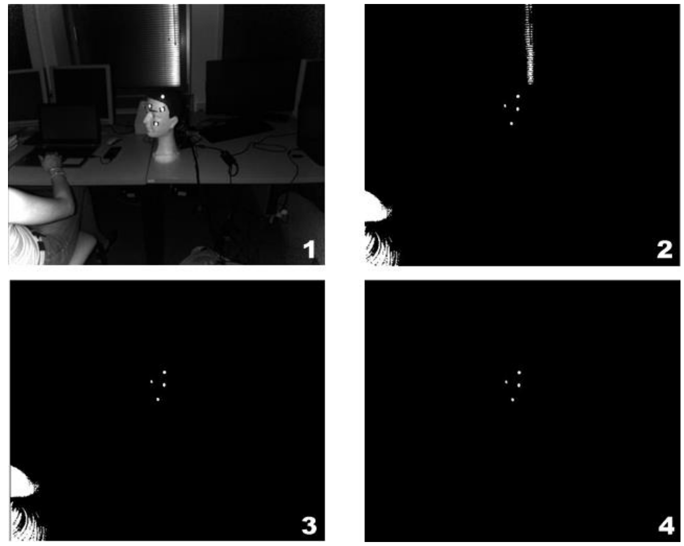
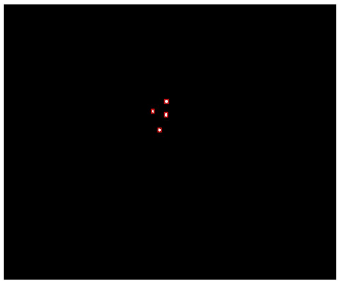


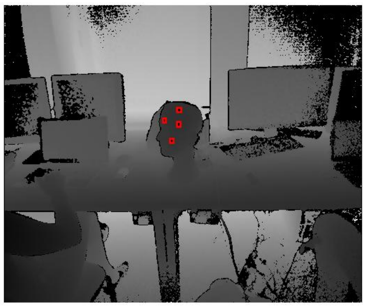
Fig. 6. The four steps of the filtering. 1: original IR image. 2: After converting to black and white. 3: Deleting all pixels which are to wide or to near from/to the camera. 4: Deleting all pixels outside of the scan area.

- The area size of the pixels
- The x and y coordinate (in pixels) of the middle of the pixel area
- The bounding box for the pixel area. This describes the smallest possible rectangle, which surrounds the pixel area, without crossing a pixel.

Now, for each cluster, the area size is checked, and if the area is in the specified region, the cluster is identified as a fiducial. As soon as the fiducial is identified, the x and y coordinates are stored in a vector. Because of the fact, that the depth image holds no information for the exact location of the fiducials, depth is calculated by the mean value of the depth of the pixels, which surrounds the fiducial. To simplify this process, the bounding box is used, to get the pixels from the depth image, which are outside of the fiducial, but close to it (see Fig. 7b). In the last step, from measuring the position of the fiducials, the x and y component of the fiducials are converted from pixels to millimeters with the help of the intrinsic matrix. Until this point, we have on the one side the reference model and on the other side normally four tracked fiducials, each with their x, y and z components. But we dont know, which tracked fiducials corresponds to which fiducial in the reference model. To calculate the translation and rotation of the tracked object relative to the camera, we have to identify each tracked fiducial with respect to the fiducials in the reference model. This is done by calculating the distances between all points (this is done with the reference model and with the tracked fiducials). To avoid problems with measuring errors, distances in the tracked system, which are to similar are sorted out. To



(a) Pixels in clusters. Red: Boundingbox of each cluster



(b) Depth image with bounding boxes around the fiducials.

Fig. 7. Identification of the bounding boxes of the fiducials.

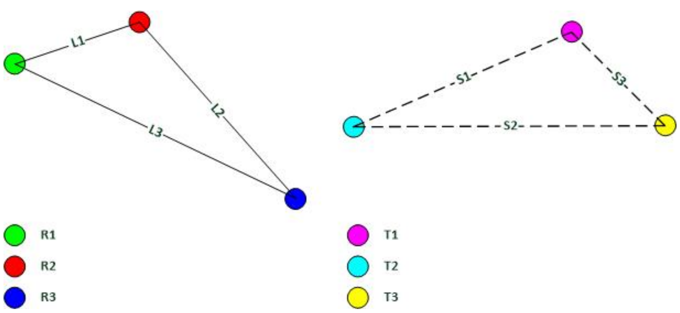


Fig. 8. Reference model (left) and tracked fiducials (right)

identify all fiducials, at least 4 distances have to remain. By comparing the distances between the tracked fiducials and the fiducials in the reference model, we can create connections between the points and identify the fiducials. This is shown in Fig. 8 and Fig. 9 (but with only three fiducials and therefore three distances). For each distance from the reference system, the matching distance from the tracked system is searched. After that, a solution is searched which holds for each point of the reference system a point of the tracked system in a way, that the matching distances are made from the same points. The algorithm is pictured in Fig. 9. The first point from the reference system (green) is taken and it is stored, which points from the tracked system it can be (cyan and yellow). Then the algorithm searches for the next distance, where the point one (green) appears. Then it checks if the yellow or the cyan point appears in the distance from the tracked system. The point which appears, corresponds to the first point. This is done for all three points from the reference system. The last process is to generate the transformation matrix out of the tracked points. To define a translation and rotation of the object, we first have to define a coordinate system for the object. The coordinate system is based on the points from the reference system, where the first point is used as base. The three unit vectors are created with the formulas below:

$$\vec{e}_x = \frac{\vec{p}_2 - \vec{p}_1}{\mid \vec{p}_2 - \vec{p}_1 \mid} \tag{9}$$

$$\vec{s} = \langle \vec{e}_x, \vec{p}_3 - \vec{p}_1 \rangle \vec{e}_x \tag{10}$$

$$\vec{e}_y = \frac{\vec{p}_3 - \vec{p}_1 - \vec{s}}{\mid \vec{p}_3 - \vec{p}_1 - \vec{s} \mid} \tag{11}$$

$$\vec{e}_z = \frac{\vec{e}_x \times \vec{e}_y}{\mid \vec{e}_x \times \vec{e}_y \mid} \tag{12}$$

This simple definition makes the calculation of the transformation matrix for the tracked points really easy. By defining the point $\vec{p}_1$ from the tracked system as base, the same formula can be used for the tracked points as it was used for the reference system.

V. Motion Compensation Algorithm

The contents of the previous sections have been combined to generate a motion compensation algorithm, which is running in MATLAB. This algorithm can be divided into two phases. The first phase is the *initialization phase*. In this phase the following steps are executed.

1) Set up the interfaces between MATLAB and the UR3-robot and between MATLAB and the tracking system.
2) Configure the parameters of the robot and of the tracking system
3) Execute the hand-eye-calibration method to determine the homogeneous transformation matrix from the tracking system to the world frame, denoted with Y and the homogeneous transformation matrix from the coil to the end-effector, denoted with X

After the initialization phase the *compensation phase* starts. This phase consists of an infinite while loop, where the following steps are executed, permanently:

1) Track the position of the fiducials, attached to the head, with the tracking system
2) Use this data to determine ${}^{\mathrm{S}}T_{\mathrm{H}}$ (see Fig. 1). This step is only necessary, if the KINECT is used

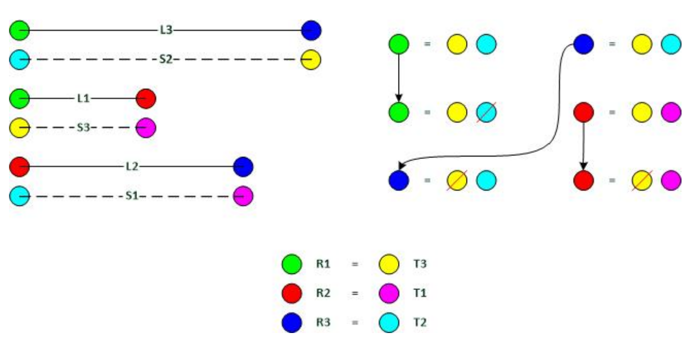


Fig. 9. Algorithm for identifying the points from the tracked system

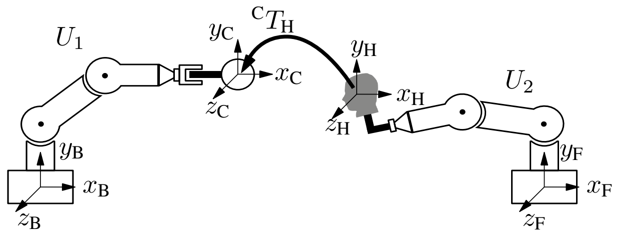


Fig. 10. Experimental setup for the validation of the motion compensation method.

3) Determine the posture of the end-effector relative to the robot's base, that the robot must attain, when the desired posture between coil and head should be maintained. The former is denoted by the homogeneous transformation matrix Z and the latter by the homogeneous transformation matrix $^{\mathrm{C}}T_{\mathrm{H,g}}$
4) Demand the actual $^{\mathrm{B}}T_{\mathrm{M}}$ from the driver program of the robot.
5) If the deviation between $^{\mathrm{B}}T_{\mathrm{M}}$ and Z is above a specific threshold, activate the path planning algorithm to determine a safe path to the desired posture in the operating space.
6) Activate the inverse kinematics subroutine to translate this path into the corresponding one in the joint space.
7) Move the robot to the desired posture.
8) Go to 1)

The homogeneous transformation matrix Z is calculated with the formula

$$Z = Y * {}^{\mathrm{S}}T_{\mathrm{H}} * {}^{\mathrm{H}}T_{\mathrm{C,g}} * X^{-1} \tag{13}$$

Note, that in the current version of the motion compensation method the path planning algorithm from step 5 is not existent, yet. Instead, a joint configuration, which corresponds to Z, is calculated with the inverse kinematics subroutine and then, all joints variables are driven to their specified values on a point-to-point trajectory. Currently, there is also no collision detection enabled. However, the motion compensation algorithm can be augmented with these two features.

VI. Experiment and Conclusion

An experiment is executed, as an indicator of the quality of the above presented motion compensation method. The setup of the experiment is shown in Fig. 10. The end-effector of the robot U_2 is holding a *head phantom*. There are markers attached to this head phantom, such that the tracking system can determine its posture. The robot is a UR5-robot. The robot U_1 carries the coil with his end-effector and the relative transformation between head and coil is denoted by $^{\mathrm{C}}T_{\mathrm{H}}$. Note, that in this experiment the robot U_1 is a *UR3-robot*, but this does merely alter the link lengths which have to be inserted into the forward and inverse kinematics. The kinematic structure of UR3 and UR5 are identical and this means that the motion compensation method is still valid. In the experiment, the robot U_2 is moving its end-effector on a trajectory, which is unknown, priorly. The motion compensation method is running and connected to robot U_1. It shall compensate the movements of the end-effector of U_2, such that Eq. (1) holds. If the matrix $^{\mathrm{C}}T_{\mathrm{H}}$ is known, then also the homogeneous transformation matrix $^{E_2}T_{E_1}$ between the end-effectors of the two robots can be calculated. The end-effector of robot U_1 is denoted by E_1 and that of U_2 by E_2. This is possible, because during the hand-eye-calibration procedure of the two robots, also the homogeneous transformation matrices between the coil and E_1 and between the head phantom and E_2 are determined. Then the matrix $^{\mathrm{C}}T_{\mathrm{H,g}}$ has a corresponding homogeneous transformation matrix $^{E_2}T_{E_1,\mathrm{g}}$, which expresses the desired transformation between the two end-effectors. In the experiment it has the value

$$^{E_2}T_{E_1,\mathrm{g}} = \begin{bmatrix} 0.8725 & 0.159 & 0.4619 & 9.5626 \\ -0.4253 & -0.2179 & 0.8784 & -160.5035 \\ 0.2404 & -0.9629 & -0.1225 & 412.4959 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

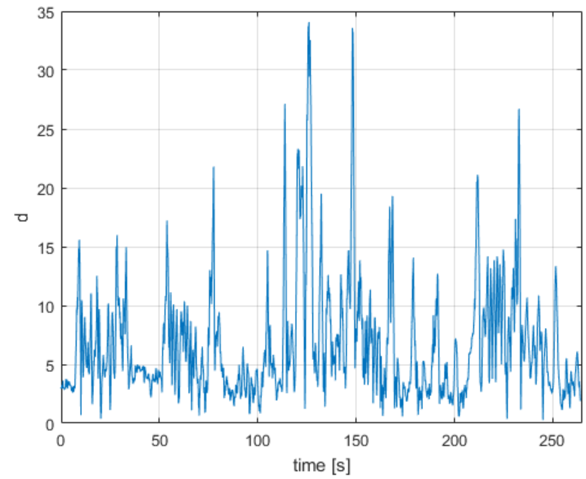


Fig. 11. Frobenius norm of the difference of $^{E_2}T_{E_1}$ and $^{E_2}T_{E_1,\mathrm{g}}$

There is a reason why, the transformation between the end-effectors is considered: During the experiment, rather than measuring $^{\mathrm{C}}T_{\mathrm{H}}$ and comparing it to $^{\mathrm{C}}T_{\mathrm{H,g}}$, the transformation $^{E_2}T_{E_1}$ is measured and compared to $^{E_2}T_{E_1,\mathrm{g}}$. The validation of the motion compensation method will be based on the deviation of $^{E_2}T_{E_1}$ from $^{E_2}T_{E_1,\mathrm{g}}$. There are multiple ways to define this deviation, but in this work it is defined as the Frobenius norm of the difference of $^{E_2}T_{E_1}$ and $^{E_2}T_{E_1,\mathrm{g}}$.

$$d = \left\| {}^{E_2}T_{E_1,\mathrm{g}} - {}^{E_2}T_{E_1} \right\|_{\mathrm{F}} \tag{14}$$

The trajectory of $d(t)$ is shown in Fig. 11. As it can be seen, the deviation is not constant, but there are high peaks, which occur, when the movements of U_2 are done with higher velocity and higher position ranges. The motion compensation method generally suffers from the high computation times of MATLAB. Therefore, the same system has a low bandwidth and there occurs a significant temporal gap between the beginning of the movements of U_2 and the beginning of the corresponding movement of U_1, which is supposed to compensate it. One possible countermeasure could be the implementation of the motion compensation method into a compiled programming language, such as C++, rather in a interpreted language as MATLAB.

REFERENCES

[1] http://www.elrotec.de/index.php/produkte/ur-roboter/vorteile-der-ur-roboter, 2016
[2] http://www.gebrauchtmaschinen-haba.com, 2016
[3] http://www.zacobria.com/temp/ur5_dimensions.jpg, 2016
[4] Ryan Keating, *UR5 Inverse Kinematics*, John Hopkins University
[5] Floris Ernst, Lars Richter, Lars Matthus, Volker Martens, Ralf Bruder, Alexander Schlaefer and Achim Schweikard, *Non-orthogonal tool/flange and robot/world calibration*, The International Journal of Medical Robotics and Computer Assisted Surgery, DOI: 10.1002/rcs.1427, 2012