KINECT-BASED TRAJECTORY TEACHING FOR INDUSTRIAL ROBOTS

Luis-Rodolfo Landa-Hurtado, luis.landa.h@gmail.com
Fabian-Andres Mamani-Macaya, fmamani1989@gmail.com
Harold-Rodrigo-Marcos Valenzuela-Coloma, harold.valenzuela.coloma@gmail.com
Manuel Fuentes-Mayas, mfsuantes@uta.cl
Ricardo-Franco Mendoza-Garcia, rmendoza@uta.cl

Escuela Universitaria de Ingeniería Mecánica, Universidad de Tarapacá, Campus Saucache, 18 de Septiembre 2222, Arica, Chile

Abstract. Industrial robots, also known as mechanical manipulators, were originally adopted by the automobile industry in the late 1970s for applications such as welding, painting, and assembly; among others. Industrialization levels are however evolving, and modern production lines must respond to more dynamic and specialized markets. For example, instead of welding an unique door model a thousand times, only a few hundreds may be required before switching to the next model. Quick generation of robot trajectories responding to that dynamism is therefore needed, but current approaches to do so, such as way-points marking with a teaching pendant and 3D-modeling of parts, are time-consuming. Also, hand-guided marking of way-points is limited to small-sized robots. This work proposes an approach to quickly generate new robot trajectories that requires a human operator and a Microsoft Kinect sensor, and focuses on a MIG welding robot application as a proof of concept. The sensor detects the skeleton of the operator, extracts way-points position and rotation information from one of his arms, extracts point-recording orders from the other arm, and then the robot autonomously generates a trajectory that joins these points, complying with position and orientation constraints, and proceeds to weld. The Kinect-based teaching approach seems appropriate for versatile generation of robot trajectories although sensor accuracy of 3D-points detection must be improved.

Keywords: kinect, robot, trajectory, industrial, welding

1. INTRODUCTION

Industrial robots, also known as mechanical manipulators, allowed industry for implementation of a flexible type of automation that could, for example, one day to weld and another day to paint. Although, compared to conventional industrial automation, this flexibility impacts on the efficiency of robots, their efficacy is kept. Automobile manufacturers were the first companies to adopt industrial robots in the late 1970s for applications such as welding, painting and assembly, among others (Schilling, 1990).

Industrialization levels are however evolving, and modern production lines must respond to more dynamic and specialized markets. For example, car design is sometimes guided by the idiosyncrasy of continents, and continents may be sub-divided into more specialized sub-markets, such as speed-demanding or eco-friendly users. A variety of models must then be handled by a modern production line, and so instead of welding an unique door model a thousand times, only a few hundreds may be required before switching to the next model. Quick generation of robot trajectories responding to that dynamism is therefore needed.

One current approach for trajectory generation is way-points marking with a teaching pendant where an operator moves the robot to desired positions, records them, and enters code to generate a convenient trajectory (Moe and Schjolberg, 2013). Another approach is 3D-modeling of parts where an operator creates a CAD model of the working object, imports them into a simulator that also has a model of the robot, and virtually generates a convenient trajectory (ABB Robotics, 2013a). Both of these approaches are however time-consuming. A last approach is hand-guided marking of way-points where an operator, instead of moving the robot with a teaching pendant, moves the robot joints by hand to the desired positions (Moe and Schjolberg, 2013). Although faster, this approach is limited to small robots and may be even dangerous when driving robots that work with high-currents, such as welding ones.

This work proposes an approach to quickly generate new robot trajectories that requires a human operator and a Microsoft Kinect sensor, and focuses on a MIG welding robot application as a proof of concept. The Kinect sensor allows for simply 3D perception of human movements (El-Laithy, Huang, and Yeh, 2012) (Xin et. al., 2013), and—in this work—it is used to detect the skeleton of the operator, to extract way-points from the right hand, and to extract point-recording orders from the left hand. Then, information is passed over a robot controller that, complying with position and orientation constraints, autonomously generates a trajectory joining these points and that executes this trajectory in a virtual or real robot.

The Kinect-based teaching approach seems appropriate for versatile generation of robot trajectories although measurement stability, trajectory generation heuristic, and angle of vision of the sensor for larger teaching areas must be improved.
1.1 Related work

The Kinect sensor has already been used in a variety of robotic applications, such as body-recognition for human tracking, environment recognition for simultaneous localization and mapping (SLAM), and telepresence for movement replication (El-laithy, Huang, and Yeh, 2012). Besides 3D-perception of human movements, the Kinect sensor raw information, such as images from internal RGB or Infrared cameras, can be accessed to expand its capabilities. For example, El-laithy, Huang, and Yeh (2012) mounted a Kinect sensor on a mobile robot and performed object detection and collision avoidance for indoor navigation. Their work also pointed out important limitations of the Kinect, such as unreliable depth maps when artificial or sun light was in front of the sensor, or the impossibility of detecting transparent objects. They also elaborated on the accuracy of sensor measurements, which was comparable to that of much more expensive laser sensors. To continue, Almetwally and Mallem (2013) used a Kinect to replicate human movements for tele-operation and tele-walking of a humanoid NAO robot. In this work, Kinect one more time fulfilled the functionality of a much more sensor; an expensive XSens MVN motion capture system. Finally, Moe and Schjolberg (2013) controlled an industrial robot using hands to indicate position and a smartphone to indicate orientation. Specifically, they used the accelerometer sensor of the smartphone to indicate rotation, as it was impossible to obtain such information just from the hand or, in other words, the last joint of the detected skeleton (a point in space). Since their robot, an UR5 from Universal Robot, apparently did not provide an API to easily acquire rotational data and to manipulate working frames, an important part of their work is devoted to the elaboration of transformations for interfacing operator and robot controller.

2. KINECT-BASED TRAJECTORY GENERATION

This work presents a flexible trajectory generation approach that does not require physical contact between the robot and the operator. Figure 1 shows the setup whose main components include a Microsoft Kinect sensor for the Xbox 360, an ABB IRC5 robot controller, an ABB IRB1600ID welding robot, and a PC with the following software installed: Microsoft Windows 7 SP1, RobotStudio v5.15.02, ABB PC SDK v5.12.04 (part of the ABB RAB v5.12.04 package), Visual C# 2010 Express, and Kinect for Windows SDK v1.8. The Kinect sensor captures high-level three-dimensional information about the operator standing in-front of it, the PC establish a link between the sensor information and the robot controller, and the robot controller learns new robot trajectories based on the sensor information and executes them using the robot.

2.1 Robot programming

RobotStudio is the software provided by ABB AB Robotics Products to program and simulate tasks for its robots (ABB Robotics, 2013a). Robot programs are written in a text-based programming language, called RAPID, that includes instructions of conventional programming languages, such as conditional statements (e.g., IF, ELSEIF, etc.) and loops (e.g., FOR, WHILE, etc.); but that also includes a rich set of robot-specialized functions, such as CalcJointT that returns a joint-domain configuration for reaching a particular position and orientation with the robot tool (ABB Robotics, 2013b). A RAPID program, called controller program (CP), is written and simulated in RobotStudio at the PC (see Fig. 1), and then it is loaded into a computer located inside the IRC5 robot controller (see Fig. 1). Besides controlling all the movements of the robot, the CP interacts with another program running at the PC that acquires information from the Kinect sensor, called Kinect program (KP). This interaction is based on the KP accessing two variables of the CP: sequenceFlag and VariableTarget, as shown in Fig. 3. Whilst sequenceFlag is a numerical value (double), VariableTarget is a RAPID data structure, called robtarget, that stores information about position and rotation of a robot tool, as well as suggested ranges of key robot-joint angles to achieve the aforementioned position and rotation. Hence, a robtarget, or
simply a target, is a conventional way-point, so the following sections deal with them as synonyms.

2.2 Kinect programming

A C# program, called Kinect program (KP), is written in Visual C# Express that interacts with the Kinect sensor by calling pre-compiled libraries (i.e., drivers) provided by the Kinect for Windows SDK. These drivers provide advanced image-processing functions that allow, e.g., for easy access to raw RGB-image frames or to the skeleton of an user standing in front of the Kinect (see Fig. 2). Although a variety of drivers are available for sensor interaction, Kinect for Windows were chosen because of their tested compatibility with the ABB PC SDK; the drivers required to interact with ABB robot controllers from the same program. This interaction will be explained in the next section.

![Example of the Kinect sensor capabilities: depth image (the darker, the nearer) and skeleton detection.](image)

After proper detection and initialization of the Kinect sensor, the KP allows for the execution of a group of C# sentences to handle a variety of events happening inside and in-front of the Kinect. Whilst one event may be “a RGB frame is ready”, another one may be “a detected skeleton is ready”. The group of orders handling an event is consequently called a “handler”. The handler responsible for the interaction with the robot controller, and so with the robot itself, is the one that reacts in front of the event “a detected skeleton is ready”. After a skeleton is detected, the handler extracts the 3D Cartesian coordinates of right and left hands in meters. Then, the heights of the hands are compared and, if and only if left hand is above the right one, actions are triggered (see Fig. 2). Each time this condition happens, a different action is triggered; hence, a sequence of actions is pre-programmed, as will be shown in the next section.

2.3 Kinect-to-robot link

In order to establish a link between the Kinect sensor and the IRC5 robot controller, the KP accesses the drivers provided by the ABB PC SDK installed at the PC. These drivers provide high-level communication functions that allow, e.g., for network scanning to detect robot controllers, or for getting and setting variables of a remote program running at a specific controller. Thus, at the PC side, the KP continuously polls whether the operator has raised his left hand above the right one, and, if so, it proceeds to execute different actions according to the sequence shown in Fig. 3. At the same time, on the controller side, the CP also runs a continuous poll to check whether sequenceFlag has been modified by the KP and so to trigger the proper processes. The evaluation of sequenceFlag is carried out according to the conditions also shown in Fig. 3.

2.4 Target validation and search

Each time an operator sends a “record target” command, the KP suggests the CP a new target composed of the X, Y, and Z coordinates of the right hand in mm and of a robot tool rotation of 45° in relation to a virtual vertical wall at Xmm distance from the Kinect. 45° is chosen because it is the optimal angle to approach a surface with a MIG welding tool. The CP then performs a search looking for a valid joint-domain configuration to achieve the suggested target. The search begins at the original target, keeps the suggested position, and simply rotates the vertical axis to the right and the left until a valid target is found. Figure 4 shows some examples of valid robot configurations found with this search. If no valid configuration is found, the suggested target is not recorded, and the last valid target is kept.

3. ROBOT TRAJECTORY REPLICATION

Once targets are captured by the Kinect and sent to the robot controller, a new trajectory is generated. This trajectory may run at a virtual robot controller connected to a simulated robot, or at a real robot controller connected to the real robot. The virtual controller runs at the PC and requires the execution of RobotStudio. A vast part of the work was carried out at the virtual controller because it provides a safe environment to begin developing algorithms that may fail and otherwise damage the real robot. The program developed at the virtual controller, the CP, runs without any modification at the real
3.1 Virtual Controller

Figure 5 shows the teaching and execution of an (almost) horizontal trajectory going from right-to-left of the operator (mirrored at the pictures). Although trajectory generation worked well, some issues appeared. For example, the position detection of the hand of the operator was fluctuating; $X$, $Y$, and $Z$ values were never stable at a coordinate, so it was difficult to replicate a trajectory or to make it perfectly horizontal. This fluctuation was however in the order of $\pm$ a few millimeters. Also, although initial and final targets of the trajectory were validated at the CP, it was not always possible to generate a linear trajectory joining them. This scenario frequently appeared when teaching trajectories that were normal to the $XY$ plane of the Kinect.

3.2 Real Controller

Figure 6 shows the teaching and execution of a crossed trajectory going from upper-right to bottom-left side of the operator (mirrored at the pictures). Although, the CP ran smoothly at the real controller, robot speed was reduced in order to protect the robot and the operator at this development stage. Also, a virtual wall located at $1.4\text{m}$ in front of the Kinect protected the operator from the reach of the robot; in other words, no way-points beyond $1.4\text{m}$ could be marked with the right hand. To continue, the vision angle of the Kinect was also an issue. If the sensor was located too low, higher way-points perhaps reachable by the robot could not be marked. The same happened if the sensor was located too low.
high, but with lower points. Even though CP placed the Kinect sensor in front of the robot; for safety, the experiment setup placed it at the right side of the real robot. Hence, an offset between marked way-points and trajectory generation existed. Although the displacement of the sensor could have been easily introduced at the CP code, the angle of vision of the Kinect would have limited way-point marking at the left side of the robot.

4. DISCUSSION

In order to stabilize measurements of the position of the right hand, a filter such as moving average could be used. This way, the operator would have to simply wait longer at each way-point before marking it.

In relation to the issue of the non-existent linear trajectory between two way-points, two possible solution could exist, and both have relation to target validation. Currently, target validation is performed only for the initial and final targets of a generated trajectory, and the search mentioned in Sec. 2.4 simply choose the first valid configuration found for each of them. One solution to the issue could be to try other valid configurations, that are currently ignored, and then to evaluate again whether exist or not a linear trajectory joining the way-points. A second possible solution could be to sub-divide the generated trajectory in shorter segments, and to attempt to generate many sub-trajectories instead. The latter solution would allow the tool to rotate many times, back and forth, around the vertical axis as it moves along the trajectory; although it would always keep the required welding angle of $45^\circ$ with the vertical XY plane.
A possible solution to the issue of the angle of vision of Kinect, which is only $\pm 27^\circ$, could be a matrix of sensors. Considering that the cost of the sensor (i.e., around 100USD) is several orders of magnitude cheaper than that of those of robots, and array of four Kinects covering a larger teaching area is feasible from the economical view-point. Proper algorithms considering the overlap between sensor measurements do not represent major challenges, so the solution is also feasible from the technical view-point.

5. CONCLUSION

This work proposes an approach to quickly generate new robot trajectories that requires a human operator and a Microsoft Kinect sensor, and that does not require any physical contact between them. The sensor detects the skeleton of the operator, extracts way-points position information from his right hand, extracts point-recording orders from the relative position of his left hand compared to that of the right one, and then the robot controller autonomously generates a trajectory that joins these points, complying with position and orientation constraints, and executes them in a robot. The Kinect-based teaching approach seems appropriate for versatile generation of robot trajectories although measurement stability, trajectory generation heuristic, and angle of vision of the sensor for larger teaching areas must be improved.

6. REFERENCES