Control of a UR5e robotic arm through a simulate 2D camera

Author: Mar Caballer Castells
Facultat de Física, Universitat de Barcelona, Diagonal 645, 08028 Barcelona

Advisor: Manel Puig i Vidal & Llorenç Servera Serapio
Departament d'Enginyeria Electrònica i Biomèdica

Abstract: Industry 5.0 –or fifth industrial revolution- will be focused on the co-operation between the human intelligence and cognitive computing -using collaborative robots. In this paper the UR5e collaborative robot from Universal Robots is the protagonist of an Offline Programming with RoboDK software. To give UR5e the sense of sight a 2D simulated camera is used as a previous step before using the Intel RealSense D-435i. Just 4 steps are needed to know the connection between the 2D image view and the scenario and 1 more to move the robotic arm to the user's desired position. In the present study it is described in detail how that has been achieved, including the description of the roboDK Tree Station and the Python's data flow diagrams.

I. INTRODUCTION

The word robot comes from the Czech word *robota* and means "forced labor" or "slave". Robotics, and specifically robotic arms -which have similar functions to a human arm-, are present almost in every field in our life, especially in production lines. On one hand, the rise of robotics in industry has increased productivity, reduced production time and improved production quality. Robots perform a perfectly choreographed dance. On the other hand, it is inevitable that people wonder if robots will "steal" their jobs in a few years. [1]

Humans use their senses to collect data from the environment, being sight the most important of all of them. Unlike humans, robots do not have biological senses and, for this reason, engineers must create them. Sensors are the senses of robots. In order to mimic human sight, robots can incorporate cameras that, together with computer algorithms, allow robots to process visual data from the world. Without these sensors, robots are essentially blind. This is not a problem for many robotic tasks, but for some applications sight is useful or even essential.

Although we may think that robots deprive humans of job opportunities, they make our lives easier in many ways by doing repetitive or dangerous work. As the name suggests, a *cobot* (collaborative robot) is a robot designed to collaborate side by side with human workers in a shared space. This is why robots are equipped with sophisticated sensors that can make a robot stop when an operator is too close and resume executing its tasks once the operator leaves.

This paper provides the opportunity to be one step closer to the ultimate goal, which is the control of the UR5e robot with an Intel RealSense Depth Camera D-435i attached to the UR5e's arm using RoboDK and a Off-Line Programming with Python.

The first step to reach the final goal and which is shown in this article, has been a virtual simulation with a 2D simulate camera and a UR5e library robot using RoboDK.

II. HARDWARE

A. UR5e robot

UR5e [2] is a collaborative robot arm from the Universal Robot e-Series family which has 4 members- the UR3e, UR5e, UR10 and UR16e. The 'e' in e-Series stands for empowering, ease of use, everyone and evolution. Each member of the family has a different reach and payload but they are all composed of 6 axes (which are: base, shoulder, elbow, wrist 1, wrist 2 and wrist 3 -also called "six degrees of freedom"). All joins have a 360-degree rotation except the UR3e, which has infinite rotation at the end joint.

UR5e is ideal for automating low-weight processing tasks with its 5kg payload and 850 mm reach radius designed to perform a wide range of applications such as quality control, pick & place, packaging, welding or painting. In Universal Robot + online showroom we can find end effectors and accessories designed to complement UR robots.

Universal Robots patented a Graphical User Interface (GUI) called PolyScope that allows operators move the robot's arm whenever they want and program them using the teach pendant (which is a 12" easy-to-use touchscreen with an intuitive 3D visualization). In Universal Robots Online Academy we can find free modules that help employees with no programming experience quickly set up and operate with robots. This gives an opportunity to gain first-hand knowledge about robots, therefore preventing additional maintenance expenses.

Universal Robots + and Universal Robots Academy are both in [2].

B. Intel RealSense Depth Camera D-435i

Intel Realsense D435i [3] is part of a Intel RealSense D400 series of cameras that combines the depth sensing capabilities of D435 with the addition of an inertial measurement unit (IMU). Due to its global shutter image

capture method –which means that sensors scan the entire area of an image at the same time-, the D435i is the appropriate camera for capturing moving objects.

D435i uses an RGB sensor with color image signal processing.

D435i uses stereo vision to calculate depth. Stereo vision implementation consists of a left imager and a right imager –simulating human binocular vision- and an optional infrared projector -to improve depth accuracy in scenes with non-texture, uniformity or low light. Both images are used to obtain a disparity map from which depth values are calculated for each pixel.

D435i uses a 6-axis IMU which is composed of a 3-axis accelerometer and a 3-axis gyroscope. The accelerometer is used to measure linear acceleration –in X, Y, Z direction- and it usually outputs in meters per seconds squared (m/s^2), or to measure gravity if the object is not moving. From these acceleration values we can calculate pitch, roll and yay values, which are defined as the rotation around X, Y and Z axis, respectively. The gyroscope is used to measure angular velocity about the pitch, roll and yay axis and it usually output in radians per second (rad/s).

All these features make the D-435i an ideal candidate among a wide variety of applications such as drones, robots and virtual reality [4].

For working with Intel Realsense D435i -and with the other IntelRealsense D400 series- there is the open-source Intel Realsense Software Development Kit 2.0 (SDK 2.0) which includes Intel Realsense Viewer, Depth quality Tool, Debug Tools, Code Examples and Wrappers.

III. SOFTWARE: ROBODK

RoboDK [5] is a simulator focused on industrial robot applications which enables robot programs to be created, simulated and generated outside the production environment and without the presence of the robot. That is called Off-Line Programming and it has several advantages such as the removal of production downtime and the study of multiple scenarios with the aim of preventing collisions, singularities or reachability issues.

This program offers a Graphical User Interface (GUI) to model the entire workspace and use a realistic model of the robot. In particular, RoboDK has a library of over 500 robots from 50 different manufacturers including ABB, Fran, Kuka, Yaskawa/Motoman and Universal Robots. All items in the RoboDK station –robots, tools, reference frames, objects, targets and much more- are shown in the station tree, which allows intuitive understanding and modifying item's dependencies.

To build these robot programs there is a generic way, without the need to write code, which enables basic robot programming for less experienced users. Despite this, simulations and programs can also be fully created using the RoboDK's Application Program Interface (API) and a specific programming language, as for example Python, C++, Visual Basic or Matlab. In particular, RoboDK API for Python, the programming language that is used in this article, includes the Robolink and Robodk modules. On the one hand, the Robolink module is the bridge between Robodk and Python. On the other hand, the Robodk module includes a robotics toolbox for Python.

In [5] it can be found how to download RoboDK –it is recommended to install RoboDK using the default settings in our case, since we use Python language-, documentation, tips and tricks, examples and the RoboDK forum (an active forum with a quick replies), among others.

IV. TREE STATION

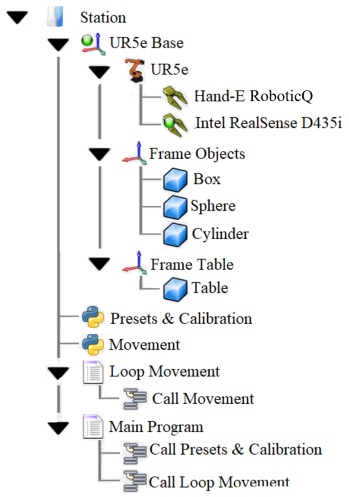

FIG. 1: Tree station.

A. Virtual robot

The UR5e robot arm has been imported from the RoboDK's online library. To find our robot we have filtered the characteristics shown in Table I. Whenever a robot is loaded in RoboDK, a reference frame representing the

Characteristics	*UR5e*
Brand	Universal Robots
Type	6 DoF
Axes	6
Payload	0-5 kg
Reach	500-1000 mm
Repeatably	0.020-0.030 mm
Weight	10-20 kg

TABLE I: Filters used to find UR5e in the online library.

robot base is added to the tree station. In the present RoboDK program, due to some functions that we are going to use later return the pose (position and orientation) with respect to the absolute reference frame, it is important that the base reference position with respect to the RoboDK station –the absolute reference- is zero in X, Y, Z, roll, pich and yaw.

B. Tools

The first tool in our station is the Hand-E Adaptive Gripper from RobotiQ [6]. However, in our work, this tool does not perform the function of gripping, but moves the Tool Center Point (TCP) into the desired position.

The second tool in our station is an Intel Realsense D-435i 3D object connected to a module which allows the fixing of the camera to the robotic arm. The camera object and the support module were designed separately by other people that have studied the physical characteristics and dimensions of the D-435i and the UR5's final effector, using a 3D design software named Autodesk Fusion 360 [7].

In this work, we use this program again to fusion these two 3D designs in order to create a unique camera & support object. To convert an object into a tool, it has been done by dragging and dropping this object (in STL format) into the robot item within the tree station.

Inside the Connect Menu section available in the Main Menu -which is located at the top of the main RoboDK's interface- you can simulate a 2D camera. It is necessary to select one item to attach the simulated camera, which in our case is the camera & support tool. Many camera parameters can be adjusted in a pop-up window, but in our case, although this procedure was useful to test the possibilities of the simulation, we configured all parameters using Python commands.

The TCP position with respect to the robot flange of each tool, can be modified with a double click at the appropriate Robot Tool in the robot Tree. In our case, the TCP positions are the middle of the end gripper's position and in the middle of the camera, respectively.

The last important thing is knowing which one is active in each moment, and we control this by using Python commands.

C. Reference Frames and Objects

The next step in our tree station design has been the definition of two Reference Frames: one for the object table –named Frame Table- and another for the objects on the table which are a box, a sphere and a cylinder –named Frame Objects.

To add a new reference frame, we select the Add Reference Frame equivalent button in the toolbar. The objects mentioned above have been imported from RoboDK's library and then they have been scaled and colored at our discretion.

D. Presets & Calibration python Program

First, the program adjusts the focal length (in mm), the field of view (in degrees), the far length -maximum working distance- (in mm) and the size of the sensor (in pixels).

Second, the program uses the path of RoboDK station to search the calibration.txt file. If the file exists, the program reads the 5 first lines (which are the pose from the previous run -calibration pose-) and, if it matches with the current pose, it means that the calibration has been done before and this first program finishes. If the file does not exist or the calibration pose does not match with the current pose, we have to calibrate. Calibration is done in 4 steps:

1. Click on one point on the simulate 2D camera view. Shown in Fig. 4.A.1.

2. Click on the corresponding point on the RoboDK scenario. Shown in Fig 4.B.2.

3. Click on another point on the simulate 2D camera view. Shown in Fig 4.A.3.

4. Click on the corresponding point on the RoboDK scenario. Shown in Fig 4.B.4.

Once this process is finished, calibration data is saved in the calibration.txt file -if the file does not exist, it is created. The flow chart of this program is shown in Figure 2.

The pair of steps 1&3 and 2&4 are defined in two different functions: Image Calibration and RoboDK Calibration. Both functions use the win32api library in order to detect the left button click. After that, Image Calibration uses a function of the same library and RoboDK Calibration uses a function of robolink library to obtain the click coordinates in pixels with respect to the left-up window corner (x_{cursor}, y_{cursor}) and in mm with respect to the RoboDK station ($x_{station}$, $y_{station}$), respectively.

The $x_{station}$, y_{cursor} station coordinates -with respect to the absolute reference- are transformed into x, y coordinates with respect to the D-435i camera (x_{camera}, y_{camera}), which allows the performance of the program in any camera orientation. To achieve this, we use the inverse camera pose matrix which corresponds to the transformation matrix -and in which both translation and rotation are taken into account- of the camera reference system with respect to the absolute system. The flow charts of both functions are in Fig. 5 and Fig. 6 in the Appendix.

The program finishes with the writing of the pose, $x^{(1)}_{cursor}$, $y^{(1)}_{cursor}$, Δx_{cursor}, Δy_{cursor}, $x^{(1)}_{camera}$, $y^{(1)}_{camera}$, Δx_{camera} and Δy_{camera} into the calibrate.txt file. The super index 1 indicates that it is the first point of calibration (steps 1 and 2) and the Δ indicates the difference between the first and the second points.

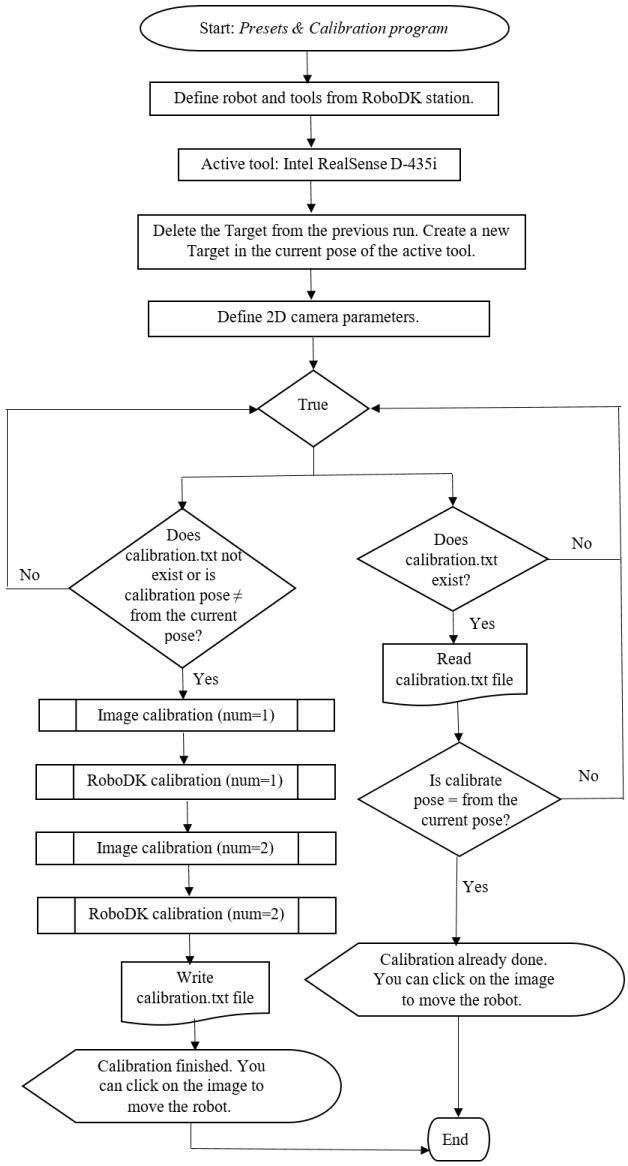

FIG. 2: Flow chart of the Presets & Calibration program.

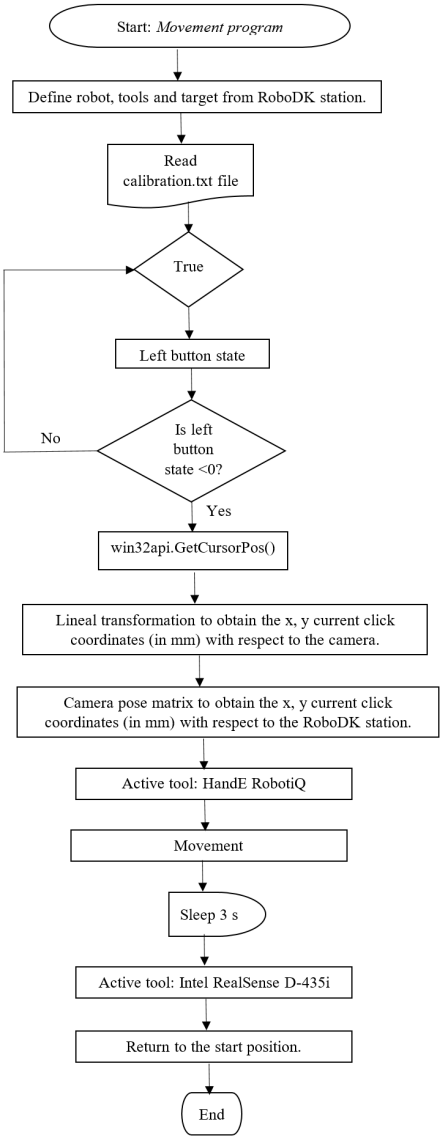

FIG. 3: Flow chart of the Movement program.

E. Movement Python Program

This program reads the calibration.txt file and when we click on the image (shown in Fig 4.A.a, Fig 4.A.b, Fig 4.A.c) -as all the calibration data are coordinates with respect to the camera (in mm) and respect to what the camera is viewing (in pixels)- once the position of the current click (in pixels) is calculated, the program does a lineal transformation to obtain the x, y current click coordinates (in mm) with respect to the camera. This is done by using the following equation:

$$x = x^{(1)}_{camera} + \frac{x_{cursor} - x^{(1)}_{cursor}}{\Delta x_{cursor}} \Delta x_{camera} \qquad (1)$$

-and the equivalent for the Y coordinate.

Finally, we use the camera pose matrix to transform the click coordinates in the camera reference system to the absolute reference system. This is the final target where the Hand-E RobotiQ moves (shown in Fig 4.C.a, Fig 4.C.b, Fig 4.C.c).

The flow chart of this program is shown in Figure 3.

F. Loop Movement and Main Program

The Main RoboDK program consists of two calls:

1. The first call is to the Presets & Calibration program.

2. The second call is to the Movement program and it runs in a loop until the end of the program when the user right clicks the window and then clicks Esc key.

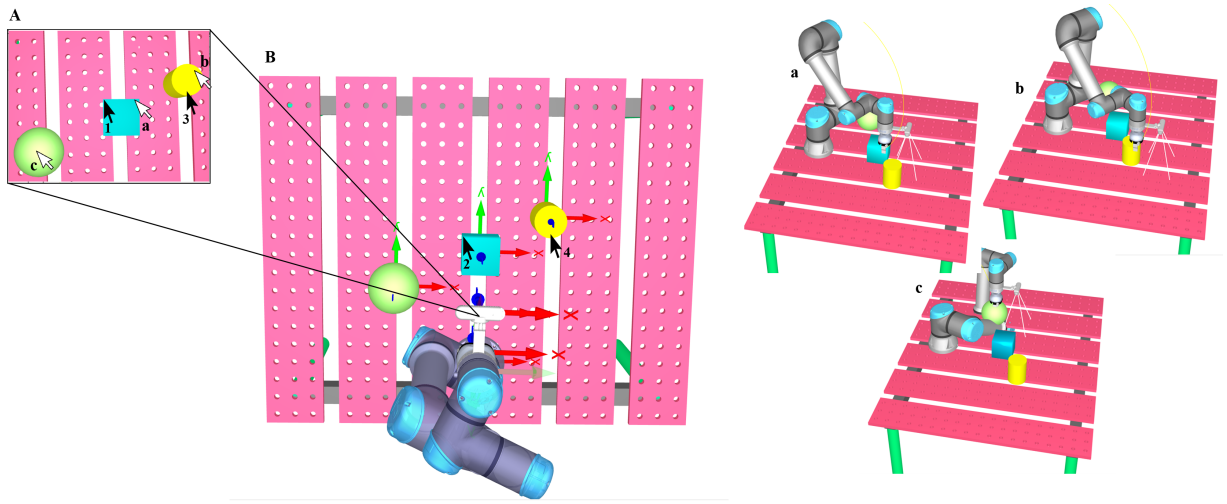

FIG. 4: **A**: Simulate 2D camera view. **B**: RoboDK scenario. **C**: Final movement in three different positions (a , b and c). A complete execution of our program is shown in this figure. First, we must follow in order the black cursor numbered sequence -1, 3 are in Fig. 4.A and 2, 4 are in Fig. 4.B- which represents the four calibration steps. Second, we must match withe cursors a, b and c -which are in Fig. 4.A- with the final movement a, b and c -which are in Fig. 4.C- respectively.

V. RESULTS AND CONCLUSIONS

A virtual simulation program using a 2D simulate camera and an UR5e library robot has been developed using RoboDK. With a simple-to-use calibration –only four clicks- you can configure the sense of sight of the robot, creating the possibility of knowing the environment –in our case a virtual scenario- and of approaching to the position decided by the user.

This offline program automatically saves data calibration, avoiding the waste of time when the program is running with the previous pose –position and orientation- of the camera. An interesting improvement is the possibility to decide if each point of calibration interests us or we want to repeat it.

2D simulated camera is limited by the absence of depth distance in each point of the scenario. In this work the depth coordinates are fixed as z = z_{camera}-100 (z is the distance with respect to the camera view, z_{camera} is the distance with respect to the absolute reference and 100 is the size of the objects in the scenario -all in mm-). With this configuration, if the camera is looking down in any direction -such as in Fig. 4- when we convert the coordinate with respect to the absolute reference, the gripper moves above the objects. But, if we have another orientation, only the x and y coordinates are right.

This study has been very helpful in guiding future work. Depth limitations shall automatically terminate with Intel RealSense D-435i and it would be only necessary to adapt this program structure to the 3D camera -with all the difficulties that come with it. The ultimate goal will provide a better sense of sight and with this, more opportunities.

Acknowledgments

I would like to express my gratitude to my advisor, Dr. Manel Puig i Vidal for his guidance and support during these months. A very special thanks to Carla for her patience and encouragement: nobody has been more important to me. Finally, I wish to thank all my family: without them, nothing would have been possible.

[1] G. Graetz, G.Michaels, *Robots at Work*, The Review of Economics and Statistics, **100**: 753-768, 2018.

[2] Collaborative robotic automation | Cobots from Universal Robots. (n.d.). [Online] Retrieved March 5, 2020, from https://www.universal-robots.com/

[3] Intel Realsense - Depth and Tracking cameras (n.d.). [Online] Retrieved March, 5, 2020, from http://intelrealsense.com/

[4] Z. Li, S. Dian,C.Li *Research on Virtual 3D Display and Teleoperation Technologies for an Inspection Robot in a Stream Generator*, In Proceedings of the 2019 4th International Conference on Automation, **57**: 1-5, 2019.

[5] Simulator for industrial robots and offline programming-RoboDK. (n.d.). [Online] Retrieved March, 5, 2020, from http://robodk.com/

[6] Start Production Faster | RobotiQ. (n.d.). [Online] Retrieved April, 10, 2020, from https://www.robotiq.com/

[7] Autodesk | 3D Design, Engineering & Construction Software. (n.d.). [Online] Retrieved April, 15, 2020, from https://www.autodesk.com/

VI. APPENDIX

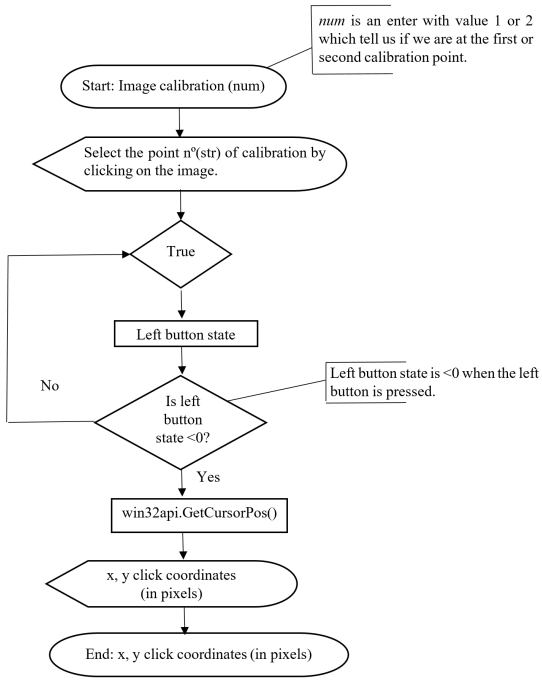

FIG. 5: Flow chart of the Image Calibration program.

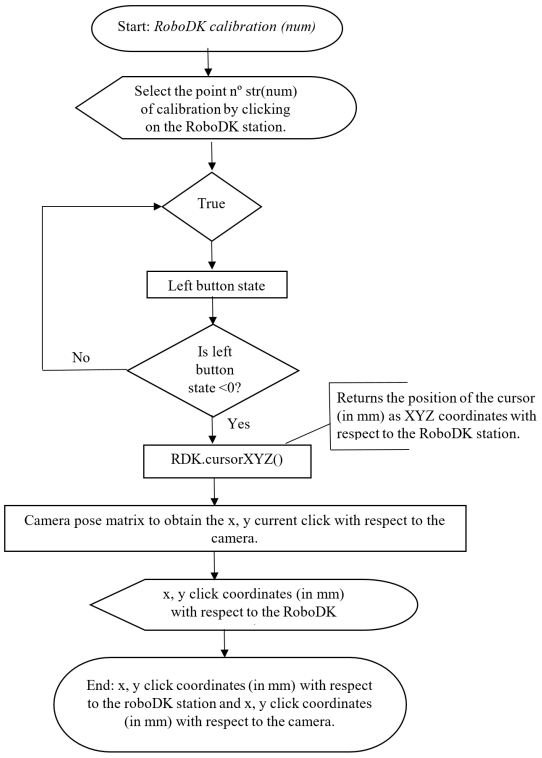

FIG. 6: Flow chart of the Image Calibration program.