Model free control of a 2DOF robotic arm using video feedback

Cristian Moldovan, Valentin Ciupe, Ion Crastiu, Valer Dolga
Department of Mechatronics,
University Politehnica Timisoara
Timisoara, Romania
cristian.moldovan@upt.ro, valentin.ciupe@upt.ro, ion.crastiu@upt.ro, valer.dolga@upt.ro

***Abstract*— In this paper we present our approach on solving the positioning problem for the Tool Center Point (TCP) of a 2 Degrees of Freedom (DOF) robotic arm similar to the way a human solves positioning tasks. The presented method relies on visual feedback and angular positions of the elements and eliminates from the equation the need to know the elements lengths, contrary to classical robotics where positioning is done through the direct and inverse kinematics method. The paper presents hardware, software and the general architecture of the system and also the proposed positioning algorithm. The scope of this research is to expand the area where robotics can be used, toward unstructured environments, where with help of vision systems, robots can successfully complete tasks.**

Keywords—machine vision, positioning, robotics.

I. INTRODUCTION

Robotics is quickly becoming a ubiquitous field especially in the manufacturing industry. With the increasing shortage of labour workforce, robotics is considered as a viable alternative to fill the gap that the industry is exposed to. Since the introduction of the first industrial robot in the late 1950's robotics has come a long way. In the early days industrial robots were bulky with big controllers and limited capabilities, but technology evolved and industrial robots became more agile, slim and also controller size started to decrease [1].

Along with the decrease in size of industrial robots came the "invasion" of robots in other fields, but the main way in which robots are used remained the same, robots were used in very structured environments. A structured environment is an environment where the position of the robot is known accurately, and is generally fixed, and also the position and orientation of the work piece is known to a great extent making it possible to pre-program the pose of the robot. In the last 2 decades there can be observed o shift in the use of robots towards unstructured environments mainly because of advances in computational power, vision systems and artificial intelligence (AI). This shift comes with the need to adapt or evolve new strategies of control for the robot [1].

In [2] there is presented a method to achieve positioning of the TCP using Deep Reinforcement learning and Transfer learning from a simulated to a real environment. The learning phase takes about 50,000 steps (about 4 hours). This method is based on training multiple Neural Networks to achieve the task. In comparison, our method does not use a Neural Net as a classifier and regression tool, instead it uses a Nearest Neighbour approach with some additional steps.

Another paper that approaches a similar subject is [3] where the authors divide robotic manipulation into two zones, the first, when robot-environment interaction can be modelled analytically i.e. interactions with rigid objects and second, interactions with soft or deformable objects where data driven models are more practical. They propose a method in which a robotic arm manipulates a rope from a starting to a target shape using learned rather than pre programmed policies.

In [4] the authors show a novel concept for kinematic-free control of a robot arm. It implements a robot controller that does not use encoder information on any joint angle and does not require any prior knowledge about the robot kinematics or dynamics. The approach works by generating actuation patterns and recording their effect on the robot's end-effector using an external camera, thereby building a local data driven kinodynamic model of the robot. The experiments with this proof-of-concept controller show that it can successfully control the position of the robot. Although this proposal is similar to the Visual Servoing [5][6] approach as system architecture, especially the use of exteroception (external sensing - e.g. a camera) in order to observe the robot's motion, they are also different in the way information is processed. For visual servoing, camera information is used to calculate the desired velocity of the end-effector and then sent to a conventional velocity controller that still uses the joint encoders to execute the motion [5], [6].

II. SYSTEM DESIGN

To control the 2DOF arm we designed an integrated system that consists of the arm, video sensor and controller PC (Personal Computer) as physical parts. The general setup is presented in Fig.1.

The kinematic structure of the arm is presented in Fig.2; we have chosen a serial structure with direct actuation for both joints. Based on the proposed kinematic structure we created a CAD model using CREO v5, Fig.3. In Fig.4 we present a

This work was supported by research grant GNaC2018 - ARUT, no. 1364/01.02.2019, financed by University Politehnica Timisoara.

978-1-7281-2906-8/19/$31.00 ©2019 IEEE

picture of the final, constructed 2DOF arm.

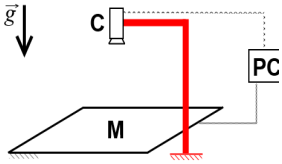

Fig.1. General setup of the system

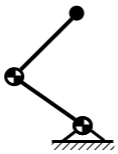

Fig.2. Kinematic structure of the 2DOF robotic arm

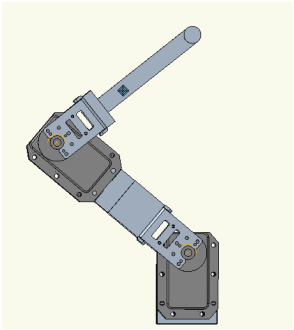

Fig. 3 CREO CAD model of the 2DOF arm

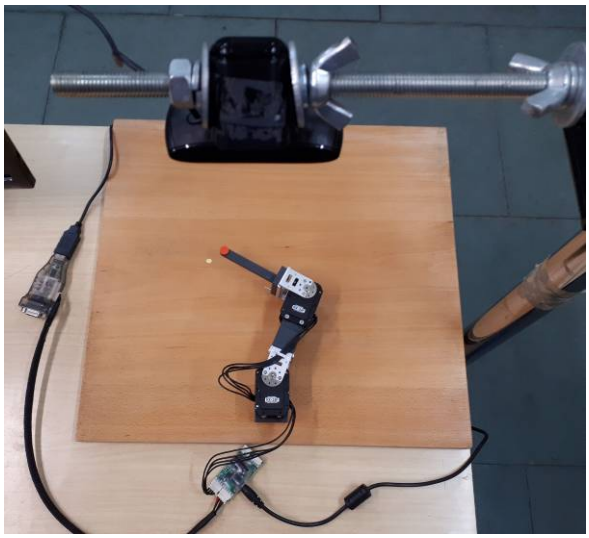

Fig.4 The System setup

For the actuation of the 2 DOF arm proposed in this paper we use 2 Dynamixel RX24F servos with an angular resolution of 0.29 degrees, rotational speed of 126 rpm and stall torque 2.6 Nm (at 12V, 2.4A) [7].

To control the 2 servos we use the Dynamixel SDK in a Python environment. The two servos are connected on a Daisychain RS485 Asynchronous Serial Communication (8bit, 1 stop, No parity) Network. The PC is connected to the servos through a proprietary hardware adapter type USB2Dynamixel.

In order to implement visual feedback for our system we have chosen as a video sensor a Logitech B910 camera running at 640x480 resolution with 30 Frames per second (FPS). The image processing is done with OpenCV and Python.

As controller for the system we use a Desktop PC with an i7-8700 processor to which both arm and video sensor are connected through USB. The PC has USB3 technology, the USB bus is capable to operate at 5gbps, so data flow is not a bottleneck, additionally, the system is not supposed to run real-time applications at this stage.

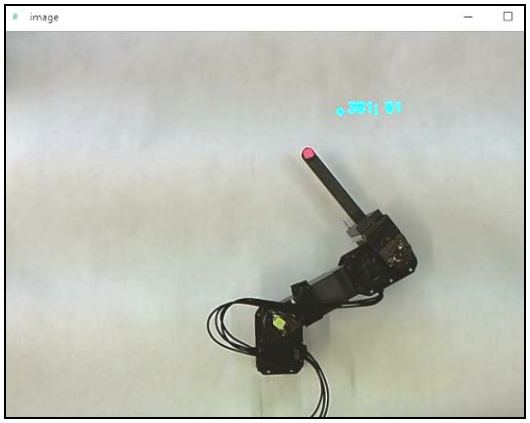

Fig. 5 System's HMI

To be able to communicate with the system, we designed a Human-Machine Interface (HMI) that allows us to specify the goal position by clicking in a window that shows the top view on the robotic arm. The HMI is shown in Fig. 5 together with the coordinates of the goal position.

III. Control Strategy

For the control of the 2DOF robotic arm using video feedback we propose a strategy that is different to traditional analytic approaches based on direct and inverse methods. Our approach relies on the data gathered by the system in order to accurately position the TCP, and works similar to the procedure a human solves positioning tasks [9].

The initial assumption is that the system only knows its current position and the goal position, this translated to a human means that he knows where it is and where it has to go. Fig. 5 shows a configuration where the arm's TCP, marked with the red marker, is at the starting, random position and it has to reach the goal position, marked through the blue point and coordinates, by clicking in the HMI. The current position of the TCP is determined by isolating the red color of the marker and determining its centroid, the goal position is inputted by the user.

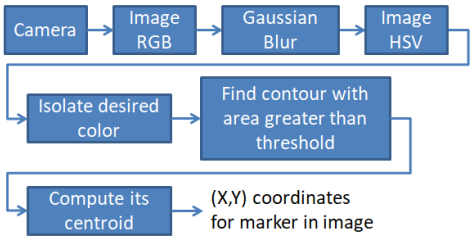

Fig. 6 Image processing steps to determine marker coordinates

The diagram in Fig.6 presents the flow of information in order to determine the marker coordinates in the image. The image from the camera is received in a RGB (Red Green Blue) format then a Gaussian Blur Filter is applied to filter noise. After this the image is converted into a HSV (Hue Saturation Value) format which is suitable for color separation. Now we can isolate the desired color in the image, more precisely the marker color. To further reduce noise and false positive areas we impose that only contours with surface area greater than a threshold value are taken into consideration to be analyzed, and since the area of the marker is larger than the noise area, the marker detection is robust. Finally we compute the centroid of the found contour which corresponds to the centroid of the marker, which in its turn is the centroid of the TCP and obtain its (X, Y) coordinates in the image.

In order to implement the above described steps we use OpenCV functions such as *GaussianBlur* – to blur the image, *cvtColor* – to convert the image from RGB to HSV color space, *findContours* – to find contours in the image, *moments* – to determine the centroid of a specific contour.

Given only this information, the system has to figure out a way to position the TCP at the goal position. For the current position, the system is also able to read the encoder values for the angular position from each servo. The Dynamixel servos are capable of providing to the system position, velocity and torque information through the specific USB connection.

To summarize, the system knows the current position of the TCP in the image $P_C\left(x_{P_C}, y_{P_C}\right)$, the corresponding θ_{1C}, θ_{2C} angular encoder values, also the goal position $P_G\left(x_{P_G}, y_{P_G}\right)$ and the system needs to determine θ_{1G}, θ_{2G} that puts the TCP at the goal position

Our approach works in 2 steps, learning and execution. In the learning phase, the arm explores its environment and records the resulted poses. By recording we mean that the TCP position in the image frame $\left(x_R, y_R\right)$ and also the angular values of the encoders θ_{1R}, θ_{2R} for that position, resulting a tuple consisting of $\left[x_R, y_R, \theta_{1R}, \theta_{2R}\right]$. In the learning or exploration phase there can be recorded a number of n arm poses resulting in a table consisting of n lines of tuples such as Table 1. We call this table "Learned positions table" or LPT.

The learning or exploration phase can be done by allowing the arm to move either randomly or systematically exploring the workspace. One can choose for example for axis 1 that has a range of 0-150 degrees, an angular step of 30 degrees resulting in 6 positions and for axis 2with range of 0-220 degrees with an angular step of 20 degrees resulting in 12 positions. The final number of generated positions can be calculated by multiplying the number of individual positions, obtaining a total number of 72 positions that are recorded in the workspace. At the end of the learning phase, the system has memorized a table with positions for the TCP in the image and the corresponding angular values of the servo-encoders that lead to the learned position. These examples are stored into a file.

TABLE 1 – RECORDED TABLE EXAMPLE

x_{R_1}	y_{R_1}	θ_{1R_1}	θ_{2R_1}
x_{R_2}	y_{R_2}	θ_{1R_2}	θ_{2R_2}
...			
x_{R_n}	y_{R_n}	θ_{1R_n}	θ_{2R_n}

The second phase is the execution phase. Here, the user specifies the goal position by clicking on the image. The goal position $P_G\left(x_{P_G}, y_{P_G}\right)$ is recorded (by clicking in the image) and a 1-Nearest Neighbours search is conducted in the LPT using the goal position as key. Now we know the tuple with the coordinates of the closest point to the goal but also the corresponding angular values for the servos, which are fed to the robotic arm in order to move to that position (the closest to the goal). We designate this point $P_C\left(x_{P_C}, y_{P_C}\right)$.

Assuming the goal position is not reached, we need to actuate the servos such that the TCP is at the goal position.

To determine the actuation commands we start at the configuration in Fig. 7 and conduct a local search algorithm.

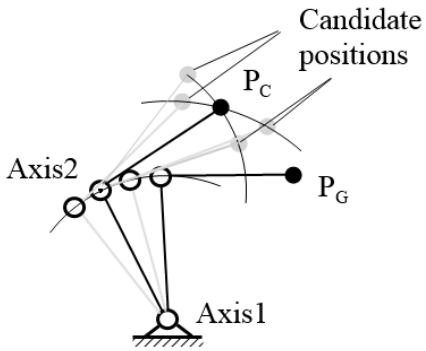

Fig.7. Local search setup with candidate positions to reach the goal

The local search algorithm consists in actuating each servo axis with α degrees clockwise and counterclockwise and determine which new position brings us closer to the goal position. The distance metric is the simple Euclidean distance computed between the candidate points and the goal point as shown in (1).

$$P_G P_C = \sqrt{\left(x_{P_G} - x_{P_C}\right)^2 + \left(y_{P_G} - y_{P_C}\right)^2} \quad (1)$$

The process is repeated until the goal position is reached with an acceptable positioning tolerance specified by the user. Since we use local search, in an area of approximate 100x100 pixels, geometric aberrations are neglected

The algorithm was implemented in Python and OpenCV. The TCP's position was obtained by placing a marker on it

and using image processing.

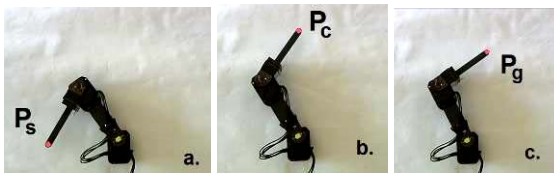

Fig. 8 Motion sequence of 2DOF arm from start to goal position

To demonstrate the process, we present in Fig.8 a sequence of images that are relevant in the TCP positioning. The robot starts in a random configuration – Fig.8a, then the controller performs a 1-Nearest Neighbors search in the tuple table. The search is conducted to find the nearest learned point to the goal point P_G based on the (x,y) coordinates. When a solution is found, the controller reads the corresponding (θ_1, θ_2) values from the table and moves the arm in that configuration. We designate this position as $P_C\left(x_{P_C}, y_{P_C}\right)$ shown in Fig.8b, this point being close to the goal point. Now we need to move the TCP to the goal location – Fig.8c, this is where we use the local search algorithm based on Fig.7 in order to move the TCP.

The local search is conducted iteratively and stops when the TCP is positioned over the target position within a specified threshold

IV. The Algorithm Pseudocode

In this chapter we briefly present the algorithm pseudo code that we used in the Python – OpenCV software implementation. In the pseudocode used some custom functions that are described separately

The Learning phase

```
01 START
    integer t1, t2, angular_step1 = 30, angular_step2 = 20,
    integer x_TCP, y_TCP, index, k = 5, m = 11
    array tuple[]
    file LPT.TXT
02 for i = 0 to k
03   for j = 0 to m
04     t1 = i*angular_step1
05     t2 = j*angular_step2
06     set_servo1_pos(t1)
07     set_servo2_pos(t2)
08     x_TCP, y_TCP = read_image_TCP_coordinates()
09     tuple[index] = add_row(x_TCP, y_TCP, t1, t2)
10     index ++
11   end for
12 end for
13 write tuple to LPT.TXT
```

In the learning phase, the arm moves through the workspace, and for each pose, the TCP coordinates and servo angular positions are recorded and stored in the LPT.TXT file.

The Execution phase

```
01 START
    file LPT.TXT
    integer x_goal, y_goal, t1, t2, threshold=10, step=5
    integer x_TCP, y_TCP
    array tuple[]
02 read LPT.TXT to tuple
03 x_goal, y_goal = get_goal_coordinates()
04 index = search_nearest(x_goal, y_goal, tuple)
05 t1 = tuple[index, 1]
06 t2 = tuple[index, 2]
07 set_servo1_pos(t1)
08 set_servo2_pos(t2)
09 x_TCP, y_TCP = read_image_TCP_coordinates()
10 while dist(x_TCP, y_TCP, x_goal, y_goal) < threshold
11     generate_candidate_positions_and_
                    _move_to_closest__position_to_goal()
12     x_TCP, y_TCP = read_image_TCP_coordinates()
13 end while
```

In the execution phase, the system reads the learned positions from the saved file. The goal position is inputted by the user by clicking in the HMI. From the learned positions, the closest is computed and the arm is moved in that position. While the distance from TCP to goal position is greater than the positioning threshold value, it generates candidate positions and move to the candidate position nearest to the goal.

For both phases, in the pseudocode we used the following functions:

- set_servo1_pos() and set_servo2_pos() command the corresponding servo to move at the specified position
- read_image_TCP_coordinates() reads the TCP coordinates from the image taking the marker color in consideration in a process described in Fig.6
- get_goal_coordinates() allows the user to select the goal coordinates by clicking on the image, as shown in Fig.5
- search_nearest(x_goal, y_goal, tuple) returns the index position of the closest point to the goal from the tuple table
- generate_candidate_positions_and_move_to_closest_ _position_to_goal() where a local search is executed to find the nearest next position for the TCP to the goal
- dist(x_TCP, y_TCP, x_goal, y_goal) computes the distance from TCP to goal according to relationship (1)

The pseudocode for the local search procedure is as following

```
01 set_servo1_pos(t1 + step)
02 dist1 = dist(x_TCP, y_TCP, x_goal, y_goal)
03 candidate1 = [t1+step, t2]
```

```
04
05 set_servo1_pos(t1 - 2*step)
06 dist2 = dist(x_TCP, y_TCP, x_goal, y_goal)
07 candidate2 = [t1 - 2*step, t2]]
08 set_servo1_pos(t1)
09
10 set_servo2_pos(t2 + step)
11 dist3 = dist(x_TCP, y_TCP, x_goal, y_goal)
12 candidate3 = [t1, t2 + step]
13
14 set_servo2_pos(t2 - 2*step)
15 dist4 = dist(x_TCP, y_TCP, x_goal, y_goal)
16 candidate4 = [t1, t2 - 2*step]
17
18 if dist1 == minimum(dist1, dist2, dist3, dist4)
19    set_servo1_pos(candidate1[1])
20    set_servo2_pos(candidate1[2])
21 endif
22
23 if dist2 == minimum(dist1, dist2, dist3, dist4)
24    set_servo1_pos(candidate2[1])
25    set_servo2_pos(candidate2[2])
26 endif
27
28 if dist3 == minimum(dist1, dist2, dist3, dist4)
29    set_servo1_pos(candidate3[1])
30    set_servo2_pos(candidate3[2])
31 endif
32
33 if dist4 == minimum(dist1, dist2, dist3, dist4)
34    set_servo1_pos(candidate4[1])
35    set_servo2_pos(candidate4[2])
36 endif
```

V. Conclusion

We have presented our work on the development of a 2DOF robotic arm and a novel control method based on a model free approach used to accurately position the TCP.

The positioning accuracy is limited by the servo's encoder resolution and the resolution of the video sensor implemented, since our approach heavily relies on visual feedback for positioning. Throughout our experiments we achieved 3 mm positioning accuracy, a good result taking into consideration that the arm's element lengths is for both elements 100 mm.

Also, when considering repeatability, a determining element is the backlash coming from the gears inside the servos. Because of this, repeatability is influenced, but still within the threshold prescribed by the algorithm.

The time spent by the system in the local search phase can be reduced. The current local search algorithm chooses between 4 candidate positions the one that is the closest to the goal, search time could be reduced by implementing a more sophisticated algorithm that takes into account the search direction and the previous position, thus reducing the search space. As future work, it is one of our goals to improve the local search method.

References

[1] Jim Beretta, Louis Finazzo, Kevin Heath, Samir Patel - Where are Robots Going?, June 13, 2019 https://www.robotics.org/webinar-detail.cfm/webinars/where-are-robots-going/id/62

[2] Andrei A Rusu, Mel Vecerík, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, Raia Hadsell - Sim-to-Real Robot Learning from Pixels with Progressive Nets, https://arxiv.org/abs/1610.04286, 1st Conference on Robot Learning, Mountain View, United States, CoRL 2017

[3] Angelina Wang, Thanard Kurutach, Kara Liu, Pieter Abbeel, Aviv Tamar - Learning Robotic Manipulation through Visual Planning and Acting, https://arxiv.org/pdf/1905.04411.pdf, 11.05.2019

[4] P. Kormushev, Y. Demiris, D. G. Caldwell, "Kinematic-free position control of a 2-DOF planar robot arm", 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Hamburg, 2015

[5] F. Chaumette, S. Hutchinson. Visual servo control, Part I: Basic approaches. IEEE Robotics and Automation Magazine, 13(4):82-90, Décembre 2006.

[6] F. Chaumette, S. Hutchinson. Visual Servo Control, Part II: Advanced Approaches. IEEE Robotics and Automation Magazine, 14(1):109-118, March 2007

[7] Dynamixel RX24F E-manual, http://emanual.robotis.com/docs/en/dxl/rx/rx-24f/ accessed 20.07.2019

[8] Artur Saudabayev, Zhanibek Rysbek, Raykhan Khassenova & Huseyin Atakan Varol - Human grasping database for activities of daily living with depth, color and kinematic data streams, Nature Scientific Data volume 5, Article number: 180101 (2018) https://rdcu.be/bOF78

[9] A. Gasparetto, P. Boscariol, A. Lanzutti, R. Vidoni "Path Planning and Trajectory Planning Algorithms: A General Overview" Motion and Operation Planning of Robotic Systems. Mechanisms and Machine Science, vol 29. Springer, 2015 pp. 3-27.

[10] A. Biason, G. Boschetti, A. Gasparetto, A. Puppatti, V. Zanotto, "Design of a Robotic Vision System", AMST'05 Advanced Manufacturing Systems and Technology, Springer, 2005

[11] Dynamixel RX24F E-manual, http://emanual.robotis.com/docs/en/dxl/rx/rx-24f/ accessed 20.07.2019

[12] Scott C. Brown, Kevin M. Passino - Intelligent Control for an Acrobot, Journal of Intelligent and Robotic Systems, Volume 18, Issue 3, pp 209–248, March 1997