Cost-effective 4DoF manipulator for general applications[⋆]

Sandro A. Magalhães[1,2], António Paulo Moreira[1,2], Filipe Neves dos Santos[2], Jorge Dias[2,4], and Luis Santos[1,5]

[1] INESC TEC - Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência, Campus da FEUP, Rua Dr. Roberto Frias s/n, 4200-465 Porto, Portugal
[2] Faculty of Engineering, University of Porto, Rua Dr. Roberto Frias s/n, 4200-465 Porto, Portugal
[3] Institute of Systems and Robotics, Department of Electrical Engineering and Computers, University of Coimbra, Rua Silvio Lima – Pólo II, 3030-290 Coimbra, Portugal
[4] Khalifa University of Science, Technology, and Research, Abu Dhabi 127788, United Arab Emirates
[5] UTAD – Universidade de Trás-os-Montes e Alto Douro, Quinta de Prados, 5000-801 Vila Real

Abstract. Nowadays, robotic manipulators' uses are broader than industrial needs. They are applied to perform agricultural tasks, consumer services, medical surgeries, among others. The development of new cost-effective robotic arms assumes a prominent position to enable their widespread adoption in these application areas. Bearing these ideas in mind, the objective of this paper is twofold. First, introduce the hardware and software architecture and position-control design for a four Degree of Freedom (DoF) manipulator constituted by high-resolution stepper motors and incremental encoders and a cost-effective price. Secondly, to describe the mitigation strategies adopted to lead with the manipulator's position using incremental encoders during startup and operating modes. The described solution has a maximum circular workspace of 0.7 m and a maximum payload of 3 kg. The developed architecture was tested, inducing the manipulator to perform a square path. Tests prove an accumulative error of 12.4 mm. All the developed code for firmware and ROS drivers was made publicly available.

Keywords: robotic arm, manipulator control, manipulator design, position-control, 4DoF manipulator

[⋆] This work is financed by National Funds through the Portuguese funding agency, FCT – Fundação para a Ciência e a Tecnologia, within the scholarship SFRH/BD/147117/2019.
The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-030-82199-9_15

1 Introduction

Since the beginning of the research in robotics, robotic manipulators became one of the most significant research interests. Their high-capability to manipulate and grasp objects has aroused the industry's interest in using them to agile and execute repetitive tasks.

The first applications for robotic arms appeared in the industry. In the beginning, robots worked only inside closed and protected cells, executing single and repetitive tasks [1]. Nevertheless, scientists and engineers later found that these robots could work collaboratively with the human being, performing supporting tasks [2].

With good results in mobile robotics research, researchers found that mobile manipulators could be helpful in many different issues. For example, inside manufacturing, they can dynamically change the layout, execute various tasks, or transport objects between cells [3]. However, mobile manipulators' uses are not constrained to industry. Successful use cases for manipulators are visible in agricultural environments [4], medical surgery rooms [5], or consumer services [6].

Many kinds of configurations for the manipulators may be used to reach different results. They can vary in the number of joints (normally between 1 to 6 joints, but some cases of redundant configurations are frequent) and joint type, i.e., revolution (R) or prismatic (P) joints [7, ch. 1]. In agricultural environments, researchers frequently use four DoF manipulators, but there are some applications with 3, 5, or 6 DoF [8–13]. A 6 DoF manipulator normally offers higher maneuverability and dexterity but is usually expensive. On the other hand, a 4 DoF manipulator is more cost-effective, so more attractive for most agricultural tasks.

The Igus RoboLink manipulator is a cost-effective manipulator designed by Igus company[6] with 4 DoF. All the joints of the arm are revolution joints (RRRR). This manipulator has not any control hardware or software. Therefore, this paper article presents the software and hardware architecture of the manipulator. The Igus RoboLink manipulator is a commercial manipulator without the control hardware and software. So, the contributions of this paper are:

– Implementation of control hardware;
– Implementation of the position control loop[7];
– Implementation of Robot Operating System (ROS) packages to communicate with the robot, simulate it, and plan trajectories[8];
– Rough evaluation of the robot's performance.

The section 2 does an overall presentation of the manipulator, specifying some of its features. Section 3 demonstrates the forward and inverse kinematics.

[6] http://www.igus.pt/
[7] The code was made publicly available at https://gitlab.inesctec.pt/agrob/igusman4agro
[8] The code was made publicly available at https://gitlab.inesctec.pt/agrob/igusman4agro_ROS

Sections 4 and 5 present the hardware and software architecture and design considerations, respectively. Section 6 details the driver, which establishes the communication between the manipulator and ROS (Robotics Operating System) [14] through the serial port. Section 7 tests the performance of the manipulator with a fixed trajectory. Finally, section 8 presents the conclusions, identifies some found issues, and previews future work to solve them.

2 Manipulator features

The Igus RoboLink robotic arm RL-D-RBT-5532-BC in Fig. 1 is a flexible, modular, multi-purpose, and low-cost solution for manipulation and grasping. It is an articulated manipulator with 4 DoF (RRRR). Plastic worm gear with bipolar precise stepper motor equips each joint for power transmission. Moreover, to feedback each joint's position, it also has a high-resolution incremental encoder and a Hall sensor for position resetting.

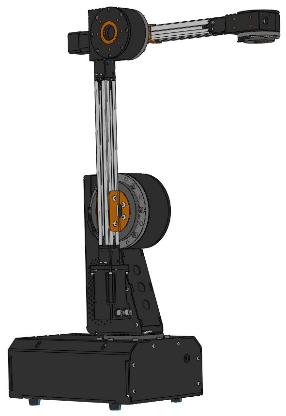

Fig. 1. Igus RoboLink RL-D-RBT-5532-BC

The total manipulator weights about 15 kg and supports a maximum payload of 3 kg. It works between 24 V to 48 V and consumes 14.6 A of maximum current. However, the manipulator is fully working with just 3 A because it has a low payload and speed in its general uses for pruning or monitoring in agriculture. Concerning the operational workspace, it reaches a maximum circular workspace of 0.7 m.

3 Kinematics

3.1 Forward kinematics

The forward kinematics of the Igus RoboLink manipulator in Fig. 1 could be calculated through the Denavit-Hartenberg (DH) method [7, ch. 3.2]. The schematics in Fig. 2, 3 and 4 present the kinematics chain of the manipulator of Fig. 1, which places the frames for each joint of the manipulator and characterise the rotation angles for each of them.

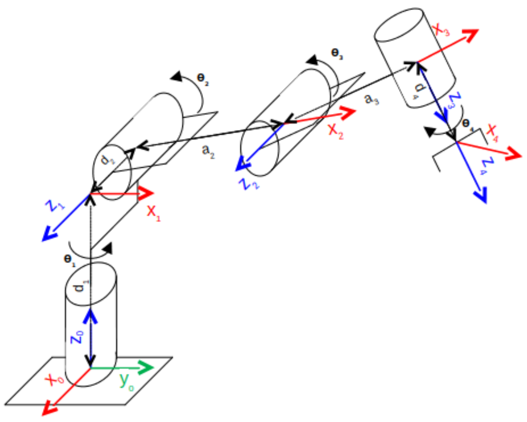

Fig. 2. Igus manipulator schematic draw for kinematics calculation. d_1, d_2, a_2, a_3 are fixed distances, respectively 0.39 m, 0.009 m, 0.36 m, 0.27 m. d_4 depends of the centre of the used gripper. $\theta_1, \theta_2, \theta_3, \theta_4$ are variables which varies according to the position of the joint. x_i, y_i, z_i are the frames for each joint i.

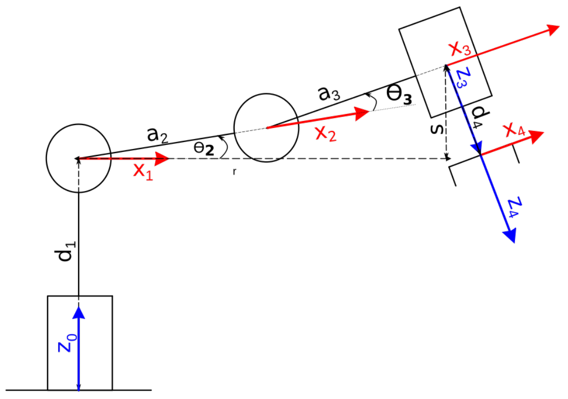

Fig. 3. Igus manipulator schematic draw for kinematics calculation (lateral view). See Fig. 2 for more details.

Any homogeneous transformation may be represented by a vector of six values: three for the rotation and three for the translation. Nevertheless, the DH

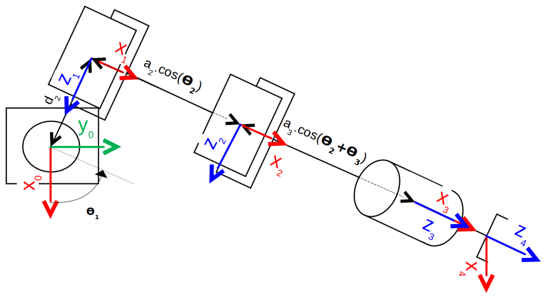

Fig. 4. Igus manipulator schematic draw for kinematics calculation (top view). The frame $\mathcal{O}x_0y_0z_0$ is the global frame, and the robot is placed on it. See Fig. 2 for more details.

method reduces this representation to four values, through careful placement of the frames for each joint [7, ch. 3.2.2], as placed in the Fig. 2. So, in the DH method, the homogeneous transformation matrix has the shape of (1). In this equation:

a_i is the segment size (distance between $\mathcal{O}_i$ and z_{i-1});
α_i is the segment twist (angle between z_i and z_{i-1});
d_i is the segment offset (distance between x_i and $\mathcal{O}_{i-1}$);
θ_i is the articulation angle (angle between x_i and x_{i-1}).

$$A_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

The table 1 applies the DH method to the Igus manipulator, using the schematics of Fig. 2. In this table, all the distances are represented in meters (m) and all angles in degrees (°), and θ_i are variable values that depend on the joint rotation.

Table 1. Application of the DH method for the manipulator

i	a_i	α_i	d_i	θ_i
1	0	90°	d_1	θ_1
2	a_2	0°	d_2	θ_2
3	a_3	90°	0	θ_3
4	0	0°	d_4	θ_4

The homogeneous transformation for each joint of the manipulator can be calculated using the matrix (1) and the table eq:forward kinematics DH. The forward kinematics of the IGUS manipulator can be computed by multiplying these homogeneous matrices (2).

$$T_4^0 = A_1 A_2 A_3 A_4 \tag{2}$$

3.2 Inverse kinematics

The forward kinematics has always been unique, but the same does not verify in the inverse kinematics. It can have one solution, multiple solutions, or no solution. The manipulator may have multiple solutions when there are multiple joint's positions to reach the same cartesian pose. There is no valid joints position when the desired cartesian pose is outside the manipulator workspace. The inverse kinematics is computed by the inverse matrix of (2), i.e., $(T_4^0)^{-1}$. Some alternative strategies are also used, such as the decoupling [7, ch. 3.3], to facilitate the inverse kinematics calculation.

Observing the manipulator's composition, it is possible to apply the kinematic decoupling strategy proposed by Spong et al. [7, ch. 3.3]. For current case, it was considered that (x_4, y_4, z_4) is the pose of the end-effector, i.e. (x_o, y_o, z_o), and (x_3, y_3, z_3) is the position of the manipulator, i.e. (x_c, y_c, z_c). Therefore, it is possible to divide the kinematic problem into two parts: (i) inverse position of the manipulator, (ii) inverse orientation of the end-effector. $\mathcal{O}_c^0$, the position of the frame $\mathcal{O}_c$ in relation to the frame $\mathcal{O}_0$, can be computed as (3). In this equation, $\mathcal{O}_o^0$ is the position of the end-effector, d_4 is the offset of the end-effector, R_4^0 is the rotation matrix between the origin and the end-effector, and because the initial pose for the manipulator is pointing up, it was considered the standard orientation of the manipulator $[0\ 0\ 1]^T$. Equation (4) decouples the orientation problem, R_j^i is the rotation matrix between the frame i to the frame j.

$$\mathcal{O}_c^o = \mathcal{O}_o^0 - d_4 R_4^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{3}$$

$$R_4^3 = (R_3^0)^{-1} R = (R_3^0)^T R \tag{4}$$

Therefore the kinematics problem can now be solved in two parts. The first part concerns computing the inverse position of the manipulator (section 3.3), computing $\mathcal{O}_c^o$, which is commonly solved through geometrical approaches. The second part concerns solving the inverse orientation of the manipulator (section 3.2), which focuses on solving the rotation matrix R_4^3.

Inverse orientation As previously referred, the computation of the inverse orientation of the manipulator focuses on solving the rotation matrix R_4^3. The rotation matrix R_4^0 is known and is the desired rotation for the end-effector. Once computed the inverse position (as performed in the section 3.3), R_3^0 can be obtained by computing T_3^0 as (2). R_3^0 is a square matrix of the first three columns and rows of the matrix T_3^0, i.e. $R_3^0 = T_3^0(0:2, 0:2)$.

While in the initial pose, the joint four's movement is a rotation along the z-axis. Therefore, the rotation matrix R_4^3, besides (4), can also be described as a matrix of Euler Angles in the z-axis (5). In this equation, α is the angle of the joint four, i.e. θ_4. Comparing both equations (4) and (5), it is possible to compute the position angle θ_4.

$$Rot_z = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

3.3 Inverse position

For computing the robot inverse position, a geometric approach was used based on the schematics of Figs. 4 and 5.

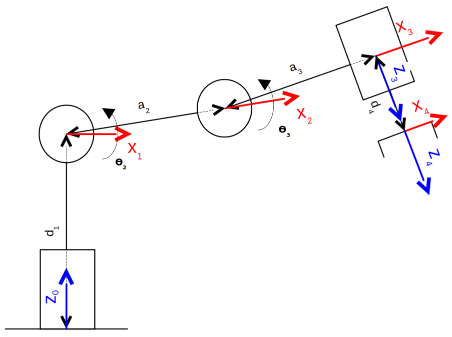

Fig. 5. Igus manipulator schematic draw for kinematics calculation (lateral view). The frame $\mathcal{O}x_0y_0z_0$ is the global frame, and the robot is placed on it. J_i is the joint i of the manipulator. See Fig. 2 for more details.

From Fig. 4, it is possible to compute the position angle of the joint 1. This angle is given by (6).

The position angles of joints 2 and 3 can be computed through a lateral view of the robot as demonstrated in Fig. 5 and are computed, respectively by (7) and (8).

$$\begin{aligned}
\theta_1 =& \arctan 2(y_3, x_3) + \\
& \arctan 2(-d_2, \pm\sqrt{r^2 - d^2}) && (6)\\
\theta_2 =& \arctan 2(s, r)- \\
& \arctan 2(a_3 \sin(\theta_2), a_2 + a_3 \cos(\theta_3)) && (7)\\
\theta_3 =& \arctan 2(\cos(\theta_3), \pm\sqrt{1 - \cos^2(\theta_3)}) && (8)\\
\cos(\theta_3) =& \frac{r^2 + s^2 - d_2^2 - d_3^2}{2 d_2 d_3} && (9)\\
r^2 =& x_3^2 + y_3^2 - d_2^2 && (10)\\
s =& z_3 - d_1 && (11)
\end{aligned}$$

4 Hardware architecture

As previously referred, the Igus manipulator (Fig. 1) is a four revolution joints manipulator. An incremental encoder counts its movement for each joint while moving through a bipolar precise stepper motor.

Due to the high-resolution of the stepper motor and once it just moves one step for each order command, a high-speed clock cycle is desired. So, in the proposed architecture, four Arduino Pro Mini[9] are responsible for commanding and managing the position of each joint (Fig. 6). A distributed system based on Arduino boards reduces the microprocessors' overload to control each joint and increase the frequency. Each microcontroller receives and processes the encoder and Hall sensor data to get the joint's absolute position. It also controls the stepper motor through a TB66600 stepper motor driver.

Since the manipulator follows a distributed architecture, the microcontrollers need to share information between them and the external systems. This communication interface is achieved using the Inter-Integrated Circuit protocol (I^2C) protocol, and the master microcontroller communicates with external systems using Universal Asynchronous Receiver-Transmitter protocol (UART) and a custom American Standard Code for Information Interchange (ASCII) string (section 5.3) as described in Fig. 6.

This distributed architecture allows the spread of the computational cost over several microcontrollers, consequently, optimises the control cycle. Simultaneously, once Electrically-Erasable Programmable Read-Only Memory (EEPROM) stores part of the data, it is also optimised, as detailed in section 5.

5 Software architecture

As referred to in section 4, the Igus RoboLink manipulator is built over a distributed configuration. Four Arduino Pro Mini are responsible for commanding

[9] https://store.arduino.cc/arduino-pro-mini

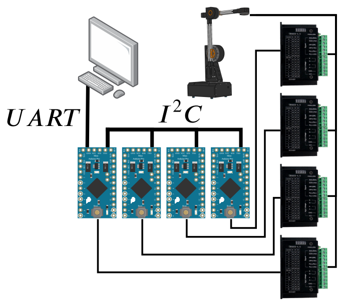

Fig. 6. Igus hardware architecture

and controlling the stepper motor drivers, which command each manipulator's joint. The microcontrollers share information using I^{2}C, where the Arduino, which controls the first joint of the robot (see Fig. 6) is the master and the other ones are slaves.

5.1 Control loop

The program's main loop is featured by reading and setting the reference and the current joint poses. In the master microcontroller, the main loop, described in the algorithm 1, starts by receiving an encoded packet through UART and decoding it. After, the master sends the reference pose to the slaves and updates its reference pose. At the end of the main loop, it requests the other joints' current poses using I^{2}C, updates its current pose, and sends an encoded packet with all updated current poses through UART.

Algorithm 1: Main loop for master (joint 1)

```
Initialization;
while true do
    Receive reference poses and velocities from UART;
    for $i = 2$ to 4 do
        Update reference pose and velocity of joint $i$ through I$^2$C;
    end
    Update reference pose of joint 1;
    for $i = 2$ to 4 do
        Request current pose of joint $i$ through I$^2$C;
    end
    Update current pose of joint 1;
    Pack all current poses and send through UART;
end
```

A similar algorithm, illustrated in the algorithm 2, is used on slaves microcontrollers. However, they do not have any UART communication and only respond to master I^{2}C requests.

Algorithm 2: Main loop for slaves (joints 2 to 4)

```
Initialization;
while true do
    Updated the transmitted reference pose;
    Respond with the current joint pose using I$^2$C;
end
```

The microcontroller processes the control loop concurrently through timer interruptions. All joints are position-controlled with a variable velocity because they have incremental encoders and stepper motors, allowing easy counting of steps and the joints' position. The program uses a control loop that simulates a Pulse Width Modulation signal (PWM) to generate the joint velocity, as described in Fig. 7. After the initialization procedures, it jumps to the STOP state. If there are a positive reference speed and a position error, the state machine commutes to PWM_HIGH. Into this state, the motor moves one step and immediately changes to PWM_LOW state or, whether the reference speed or position error is null, to STOP state. While into PWM_LOW state, the motors do not move, and the variable non_moved_steps is incremented once per control cycle of 80μs. This state machine allows a PWM effect whose duty cycle is described between $0 - 99$. So, a joint's overall speed is described in (12) in pulses per cycle. In this equation, v is the reference duty cycle between $0 - 99$.

$$v_{pps} = \frac{1}{80 \times (1 + 100 - v)} \times 10^6 \tag{12}$$

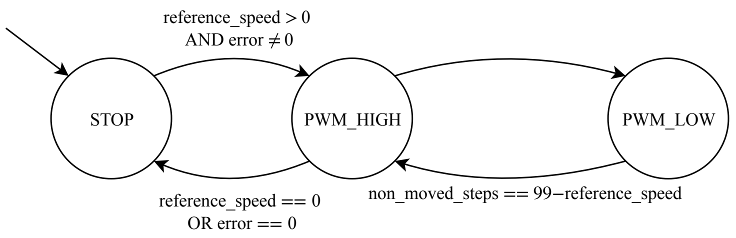

Fig. 7. State machine for joint position control using a variable velocity

The update of the joint position is done via digital pin interruption for the A pin. Fig. 8 schematics an example of the sensors' behaviour. The A and B signals allow computing the rotation direction of the incremental encoder. Whether signal A changes to true and B is true, the joint moves ClockWise (CW), and the position counter is incremented. Otherwise, the motor moves Counter-ClockWise (CCw), and the position counter decrements.

The N signal refers to the Hall sensor. When it is true, the sensor is acted, so the joint position counter is reset once.

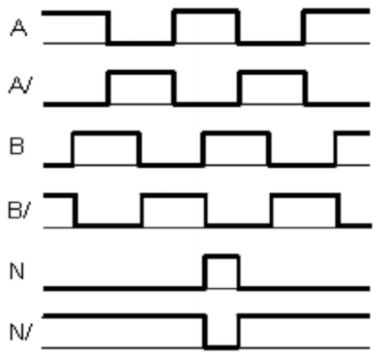

Fig. 8. Incremental encoder and Hall sensor signals

5.2 Initialization loop

There are three possible states when the manipulator powers off: stopped, moving to CW, or moving to CCw. Every time the state machine changes the state, the microcontroller stores this information in the EEPROM and the current joint position.

So, whenever the joint powers on, the microcontroller reads the joint's last state and position. Whether the joint was stopped, the last position is updated, and the initialisation procedure is complete. Otherwise, whether it was moving, the joint position needs to reset in the Hall Sensor. The joint moves to CW or CCw, such as described in the state machine of Fig. 9.

5.3 Data sharing packet protocol

As indicated in Fig. 6, the manipulator communicates with other devices through UART. For that, a structured packet for positions and velocities sharing was designed. This packet is string-based and always starts with the '$' character and finishes with the '#' character.

The equation (13) is an example of a sent packet to send the target joints positions for the manipulator. It contains eight elements of comma-separated, dividable into four pairs. Each pair contains a string of five digits representing the target position angle, in radians, for the joint multiplied by 10 000. So the joint's

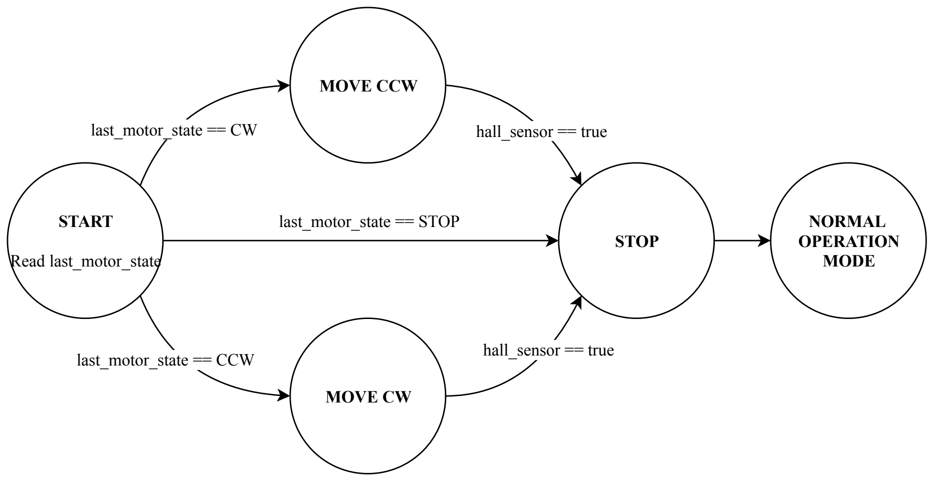

Fig. 9. State machine for joint initialization

target position should be represented by a string number between 00000 and 62 832. The other sub-string is the target velocity for Pulse Width Modulation (PWM) generation and is represented by a string of two digits between 00 and 99.

At the end of each main loop's cycle, the manipulator answers each joint's current position. This response message has a similar structure to the sent packet (14). It also starts and ends with the same characters, but it only answers four sub-strings of five digits between 00000 and 62 832.

$$\$00000, 00, 00000, 00, 00000, 00, 00000, 00\# \tag{13}$$

$$\$00000, 00000, 00000, 00000\# \tag{14}$$

In both messages, the joints are organised and numbered as indicated in Fig. 6, i.e., joint one is the base joint, and joint four is the end-effector attacher joint. When all joints are looking up and the first joint is turned to the front, the robot is in zero.

6 Driver development

A new serial driver node for ROS[10] was developed to support the communications with the external systems for further applications. This node is a simplified adaptation of *rosserial* node [15], which reads and writes string-based packets (see section 5.3) on the serial port and publishes and subscribes ROS topics.

Fig. 10 details the software architecture for the serial driver that communicates with the manipulator. When this node starts, the *SerialCommunication*

[10] See https://gitlab.inesctec.pt/agrob/igusman4agro_ROS

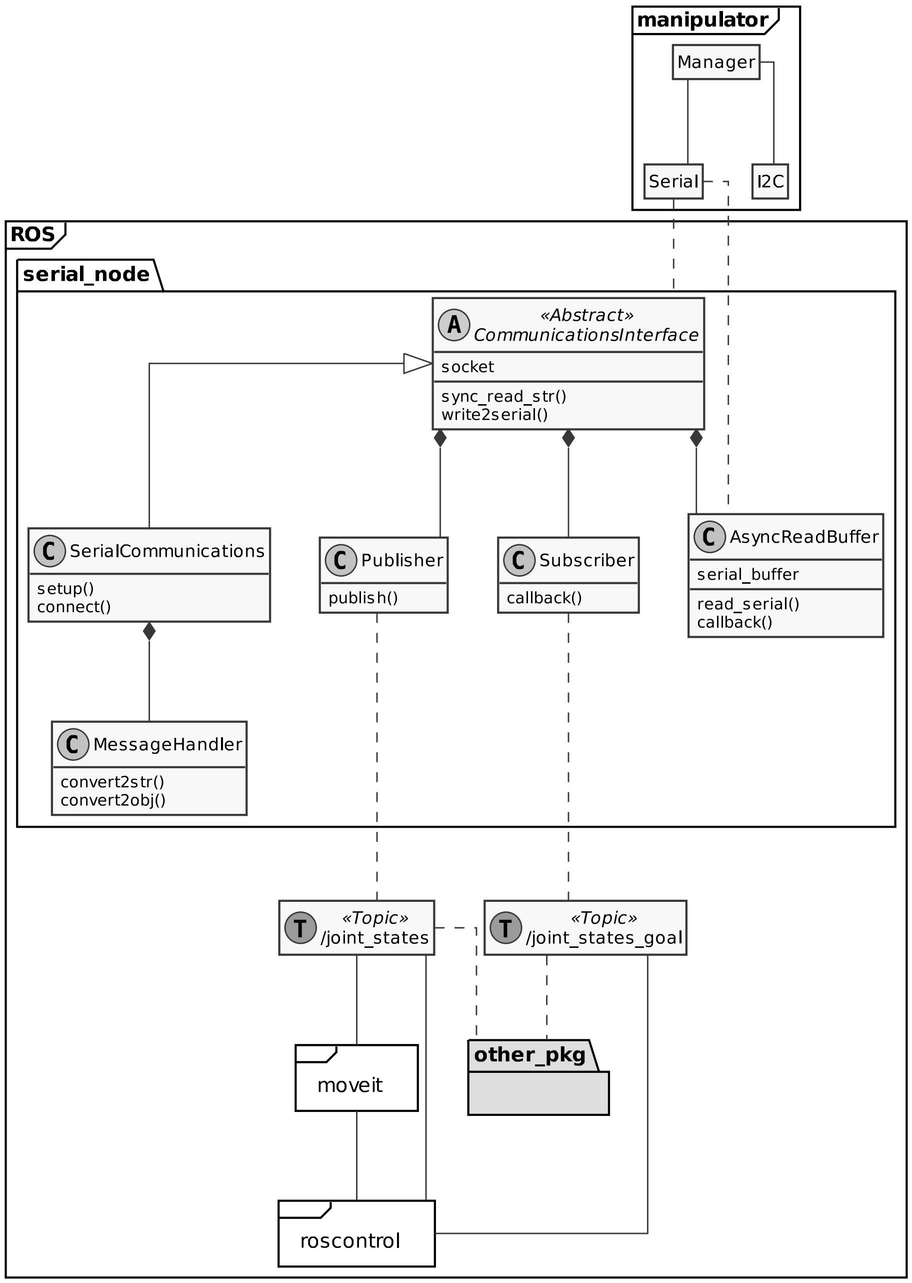

Fig. 10. UML class diagram for the ROS driver for the manipulator

class initialises the serial port and orders the Publisher and Subscriber managers' initialisation. Once the *SerialCommunication* is fully initialised, it continuously rebuilds the streamed packet, supported by *AsyncReadBuffer* class. Every time this class's object is called, it reads whole data in the serial buffer, stores it in a local buffer, and returns the required number of characters. The *SerialCommunication* asks sequentially one or more characters until a packet is rebuilt. Once the streamed string is complete, the *MessageHandler* object parses it to the Publisher publishes the manipulator's current position, encrypted in the packet, in the */joint_states* topic.

Parallelly to this execution procedure, responsible for updating the current robot position, another is responsible for streaming the target position and the desired velocity to the manipulator. So, the Subscriber subscribes to the */joint_states_goal* topic. Follow, it asks the *SerialCommunication* object to build a new string-based packet and stream it. The *MessageHandler* object supports the *SerialCommunication* ones on the string generation.

Therefore, the serial node follows an event-based runtime with two branches. The first branch reads data from the serial buffer, converts it, and publishes it on the */joint_states* topic, which reports the manipulator's joints' current position. On the other branch, the program subscribes to the */joints_state_goal* topic and generates a string-based packet written in the serial port.

Additionally, to test and work with the manipulator, some other ROS packages were developed. For instance, a configuration package for path and trajectory planning for *MoveIt!* [16] was built using the *MoveIt! Setup Assistant*. The *ROSControl* [17] is responsible for controlling the trajectory execution. Once the manipulator is position and velocity controlled, an additional contribution for the *MoveIt!* repository[11] was needed, to set up the *joint_trajectory_controllers/pos_vel_controller* and *joint_trajectory_controllers/pos_vel_acc_controller* from *ROSControl* package, which allow position, and velocity controlled robots and position, velocity, and acceleration controlled robots, respectively.

7 Experiment and results

As previously regarded, the approached manipulator is position controlled. Besides, it has many cascaded control loops to assure the correct positioning of the end-effector.

To evaluate the manipulator's performance was used a square trajectory in the YZ plane (Fig. 11). The robot moved at the maximum speed through the trajectory and did not use a path planning algorithm. The performance of the robot performing the trajectory was published online[12].

The robot could perform the trajectory with high precision, average Euclidean distance error of 12.4 mm. This error is computed as the difference between the robot's position (computed through direct kinematics) and the reference trajectory. Fig. 12 shows the computed euclidean error while the robot

[11] https://github.com/ros-planning/moveit/pull/1806
[12] See https://educast.fccn.pt/vod/clips/fqi0zj4u9/streaming.html

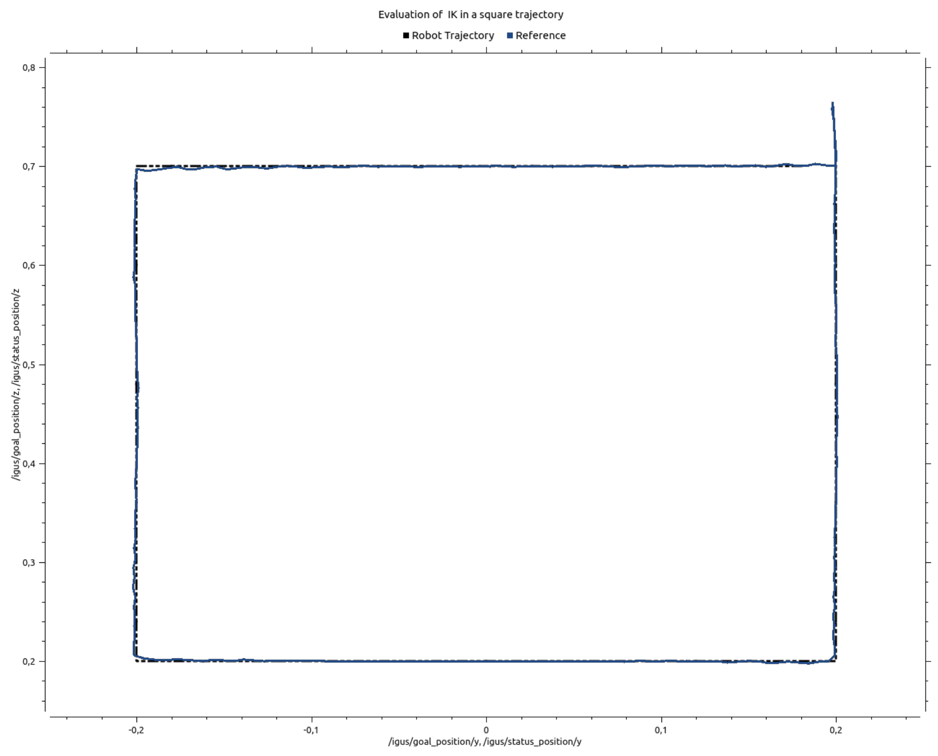

Fig. 11. Performed square trajectory by the robot in the YZ plane. Black dashed line is the reference trajectory. Blue filled line is the performed trajectory.

performs the trajectory. As shown in this figure, the robot can move easier in the z-axis than in the y-axis. Cause the similarity in the robot's behaviour between y-axis and x-axis, the same can be concluded for the last axis. Through visual observation of the robot's behaviour and analysing Figs. 11 and 12, it is observable that the robot's movement is not smooth. This robot's aggressiveness can be related to the motors' high static friction, which should be measured and balanced, or badly tuned speed or position controllers.

8 Conclusion and feature work

This paper presents an open-source and cost-effective position-controller (hardware and software) for four DoF manipulator. It moves through bipolar stepper motors, which command plastic worm gears. To feedback each joint's position, they have one incremental encoder and one magnetic Hall's sensor. This manipulator does not have the required control hardware to communicate and control the robot. Therefore a new cost-effective control circuit and algorithms were designed. The control hardware focuses on four Arduino Pro Mini boards to control each joint. The boards share data between them through I^{2}C. The boards'

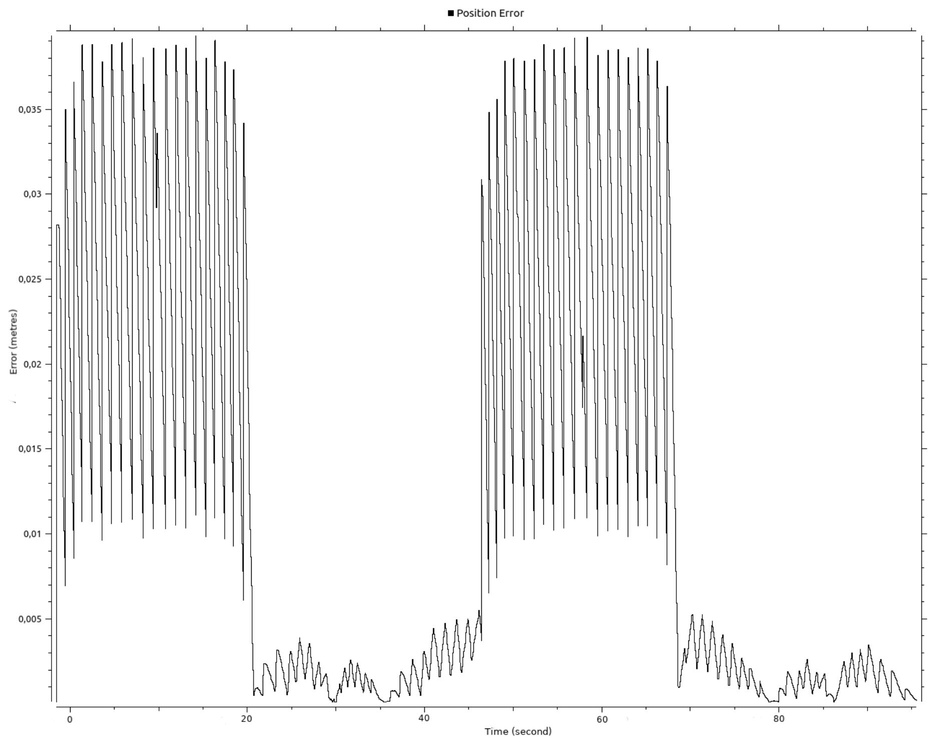

Fig. 12. Computed error of the robot to perform the trajectory.

firmware was optimised to run at 12.5 MHz, and position control the robot's joint with high-precision. The communication with external devices is made through UART.

In order to manage the platform in more complex environments, for instance, over mobile platforms, a new ROS driver was programmed. This node communicates with Moveit! to develop more complex algorithms, like advanced inverse kinematic modules, path planning, and trajectory control loops.

All the developed code for hardware control and ROS integration was made publicly available.

Due to the high resolution of the Igus RoboLink manipulator's stepper motors, they are relatively slow but precise. The usage of a distributed architecture with four Arduino boards benefits to optimise the control cycle to 80μs.

Despite their high resolution and precision, the encoders bring some issues that need to be software outcome. Once they are incremental, they cannot store their absolute position. Therefore, the developed firmware develops advanced strategies to record the joint's position at each state change, saving the EEPROM usage. However, when the manipulator shuts down during a movement, special initialisation procedures need to be used. For these particular situations, the robot recognises the last movement of the manipulator in the startup and moves it to the opposite until Hall's sensor resets the joint's position value.

A rough analysis was also performed to evaluate the performance and the functionality of the developed system. The robot could to perform the trajectory precisely but additional control improvements are required to assure a faster and smother trajectory. The robot get the trajectory with an average euclidean error of 12.4 mm.

Future work involves finding hardware and software strategies to minimise special initialisation procedures to power on the manipulator. Besides that, the quantitative measure should be done to quantify the manipulator capabilities. So, robustness, repeatability, and accuracy tests should be performed. The current implementation of the position controller focuses on a proportional controller. More advanced control loops need to be designed to assure more position accuracy and smoothness and lower overfitting.

Bibliography

[1] M. Edwards. Robots in industry: An overview. *Applied Ergonomics*, 15(1):45 – 53, 1984. ISSN 0003-6870. doi: https://doi.org/10.1016/S0003-6870(84)90121-2. URL http://www.sciencedirect.com/science/article/pii/S0003687084901212.

[2] C. Park, J. H. Kyung, and D. I. Park. Development of an industrial robot manipulator for the easy and safe human-robot cooperation. In *ICCAS 2010*, pages 678–681, Oct 2010. doi: 10.1109/ICCAS.2010.5670249.

[3] Ole Madsen, Simon Bøgh, Casper Schou, Rasmus Skovgaard Andersen, Jens Skov Damgaard, Mikkel Rath Pedersen, and Volker Krüger. Integration of mobile manipulators in an industrial production. *Industrial Robot*, 42(1):11–18, 2015. doi: 10.1108/ir-09-2014-0390.

[4] S. Kitamura and K. Oka. Recognition and cutting system of sweet pepper for picking robot in greenhouse horticulture. In *IEEE International Conference Mechatronics and Automation, 2005*, volume 4, pages 1807–1812 Vol. 4, July 2005. doi: 10.1109/ICMA.2005.1626834.

[5] N. Padoy and G. D. Hager. Human-machine collaborative surgery using learned models. In *2011 IEEE International Conference on Robotics and Automation*, pages 5285–5292, May 2011. doi: 10.1109/ICRA.2011.5980250.

[6] A. Pratkanis, A. E. Leeper, and K. Salisbury. Replacing the office intern: An autonomous coffee run with a mobile manipulator. In *2013 IEEE International Conference on Robotics and Automation*, pages 1248–1253, May 2013. doi: 10.1109/ICRA.2013.6630731.

[7] Mark W. Spong, Seth Hutchinson, and Mathukumalli Vidyasagar. *Robot modeling and control*. Wiley, 2006.

[8] Babu Sivaraman and Thomas F Burks. Robot manipulator for citrus harvesting: Configuration selection, 2007. URL http://elibrary.asabe.org/abstract.asp?aid=24066&t=5.

[9] Satoru Sakai, Michihisa Iida, Koichi Osuka, and Mikio Umeda. Design and control of a heavy material handling manipulator for agricultural robots. *Autonomous Robots*, 25(3):189–204, Oct 2008. ISSN 1573-7527. doi: 10.1007/s10514-008-9090-y. URL https://doi.org/10.1007/s10514-008-9090-y.

[10] S.S. Mehta and T.F. Burks. Vision-based control of robotic manipulator for citrus harvesting. *Computers and Electronics in Agriculture*, 102:146 – 158, 2014. ISSN 0168-1699. doi: https://doi.org/10.1016/j.compag.2014.01.003. URL http://www.sciencedirect.com/science/article/pii/S0168169914000052.

[11] Seiichi Arima, N Kondo, Y Yagi, M Monta, and Y Yoshida. Harvesting robot for strawberry grown on table top culture, 1: Harvesting robot using 5 dof manipulator. *Journal of Society of High Technology in Agriculture (Japan)*, 2001.

[12] José Nuno Gomes de Brito. *Manipulador robótico para poda automática (Projecto ROMOVI).* Msc thesis, Faculty of Engineering of the University of Porto, 2018.

[13] Sandro Augusto Magalhães, Filipe Neves dos Santos, Rui Costa Martins, Luis F. Rocha, and José Brito. Path planning algorithms benchmarking for grapevines pruning and monitoring. In Paulo Moura Oliveira, Paulo Novais, and Luís Paulo Reis, editors, *Progress in Artificial Intelligence*, pages 295–306, Cham, 2019. Springer International Publishing. ISBN 978-3-030-30244-3.

[14] Stanford Artificial Intelligence Laboratory et al. Robotic operating system, 2018. URL https://www.ros.org.

[15] Michael Ferguson, Paul Bouchier, and Mike Purvis. rosserial, 2018. URL http://wiki.ros.org/rosserial.

[16] David Coleman, Ioan Alexandru Sucan, Sachin Chitta, and Nikolaus Correll. Reducing the barrier to entry of complex robotic software: a moveit! case study. *CoRR*, abs/1404.3785, 2014. URL http://arxiv.org/abs/1404.3785.

[17] Sachin Chitta, Eitan Marder-Eppstein, Wim Meeussen, Vijay Pradeep, Adolfo Rodríguez Tsouroukdissian, Jonathan Bohren, David Coleman, Bence Magyar, Gennaro Raiola, Mathias Lüdtke, and Enrique Fernández Perdomo. ros_control: A generic and simple control framework for ros. *The Journal of Open Source Software*, 2017. doi: 10.21105/joss.00456. URL http://www.theoj.org/joss-papers/joss.00456/10.21105.joss.00456.pdf.