COBEM 2021 26th International Congress of Mechanical Engineering
November 22-26, 2021, Virtual Congress, Brazil

COB-2021-1243
DESIGN ASPECTS OF A SYSTEM FOR RESEARCH OF INTEGRATION OF COMPUTER VISION AND MACHINE LEARNING IN ROBOTICS

Carlos Rodrigues Rocha
Amanda Jorge Mendes
Kauã Ortiz Silveira
Federal Institute of Education, Science and Technology of Rio Grande do Sul
carlos.rocha@riogrande.ifrs.edu.br, amanda7rg@hotmail.com, kauaortizz@gmail.com

Abstract.*This paper presents the first results in the design and prototyping of an open source manipulator robot system with an integrated intelligent vision module. The project objective is to build a robotic cell for learning and research purposes, with a particular focus on the integration of machine learning and computer vision in robotic manipulation systems. It is based on embedded computational/microcontroller platforms programmed in Python, with different APIs for machine learning, computer vision and controlling the hardware. Initial tests showed an accuracy of approximately 99% in the classification process. This results and the prototype development are discussed here. It should also be noted that this is a final term work for an Industrial Automation Technical course.*

Keywords: *Robotic manipulation, Computer vision, Machine learning, Robotic cell, Educational platform*

1. INTRODUCTION

Robotic maniputators are widely used in industry, for reasons that vary from enhancing the production volume to avoiding exposure of humans to dangerous/hazardous environments or tasks (Romano and Dutra, 2002). Production lines use manipulators for different tasks, as pick-and-place, assembly, inspection and packaging. Usually, these robots are programmed to execute repetitive tasks in a structured environment, which is characteristic in mass manufacturing.

As industries evolve to achieve efficiency, cost reduction and sustainable production, there is need to evolve production lines as well. Flexible and lean manufacturing processes are increasingly being adopted, requiring flexible automation systems. To robots, it means that they start to work in less structured environments, where they could be working in cooperation with other robots or even humans. Traditional system architectures and programming are not well suited to these new demands, as it would require more sensory acquisition and processing to comply with tasks that require to operate in uncertain conditions and may vary rapidly. Computer vision is more and more used in these scenarios as computational power and hardware increase over time. Machine learning techniques allow to develop flexible systems that can meet the requirements of these new flexible, unstructured production lines (Piccinini *et al.*, 2009).

This demand for intelligent and collaborative robots makes the integration of computer vision and machine learning in manipulator robots a compelling research area. Its potentials and limits are still discussed (Sünderhauf *et al.*, 2018). Some research addresses the use of deep learning for task planning using digital twins, which is a key part of the Industry 4.0 paradigm (El-Shamouty *et al.*, 2019). Reliable identification by machine learning/computer vision techniques for robotics is a valid and important concern, since there are several factors that can lead to errors, even deliberated ones (Melis *et al.*, 2017). Computer vision based pick and place operations are applicable not only in industry, but also in field robotics (Wang *et al.*, 2019).

This work intends to be a contribution to this research field. Its objective is to design and prototype a fully programmable open source robotic cell to allow study and research of computer vision, machine learning and robotics, as well as their integration in production line task. To identify the design requirements, a case study was devised – a pick and place task, where different objects would be scattered over a workspace and the robot should classify them by picking every object in the workspace and placing them in the proper containers. The objects would be randomly positioned, and the robotic system should be able to correctly locate them and classify, in order to put them in the correct container.

Named RISO (acronim for Robô Inteligente Separador Organizador – Separator Organizer Intelligent Robot), the project is a work in progress, and its first results are presented in the following sections. The conceptual design phase will be detailed next, followed by the presentation of the current development of a prototype. The initial results of the computer systems implementation are discussed, followed by final remarks.

2. CONCEPTUAL DESIGN

Identification of design specifications was based on the initial case study, choosing metallic nuts and bolts as objects that should be randomly positioned in a rectangular workspace. Though it was a specific scenario, it was believed that it represented the class of pick and place tasks, being easily generalized for other types of objects. The rectangular workspace could be fixed, as a table, or a section of a conveyor belt in a production line. Different classes of objects could be manipulated by suitable tools that could be mounted on the end effector.

From the devised scenario and the notion that the system would be used for research and learning, the specifications were enlisted:

- Size and Weight: The robotic system should be transportable to be used in different scenarios, and easily mounted as well. This was important to define the size of the manipulator;
- Manipulator architecture: Pick and place tasks usually occur in planar workspaces, where an object is to be grabbed by the manipulator, lifted enough to avoid collision with other objects, transported to other position, where it is lowered and released. This motion is tipically a Schönflies displacement, which could be achieved by a 4 degrees of freedom (DOF) manipulator (Siciliano *et al.*, 2009);
- Computer vision/machine learning system: This system should use an adequate resolution digital camera, which could be fixed (relative to the workspace) or mounted on the manipulator. Cost was a concern that limited the choices of devices. Also, it was important that the image transfer to the vision processor would be fast enough. Regarding the computational aspects, hardware and software should be easily available, well documented and have full access to resources to allow implementation of different approaches regarding the proposed objective of the system – locate and classify objects in a workspace;
- Reproducible and extensible: From the start, it was considered that this project should be reproducible for other groups, and extensible, to adapt it to other scenarios. This specification affects all the others, since it has impact on production costs and choices of software/hardware. This lead to the open source feature of this project, in order to contribute to the community of researchers and students of robotics.

These specifications were analyzed and a conceptual system was proposed, as illustrated in Fig. 1. This simplified diagram shows the robotic cell components and the connections between them. The system is composed by a serial manipulator and a fixed camera with lighting system.

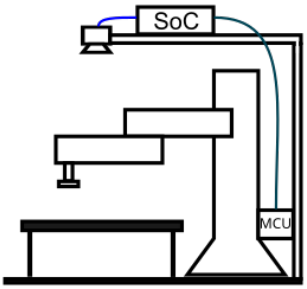

Figure 1. Simplified representation of the robotic cell.

The camera is positioned above the workspace and the manipulator. In this way, a single camera can be used to acquire images of the workspace and of the robot at the same time. Though the arm could block the view of some of the objects in a given moment, it was considered that in the initial stage of the task, the arm would be in a configuration that allows full view of the workspace, and the objects do not move during the task. It is a simplification for the initial developments. A SoC (System on a Chip) board is used as the computer vision and machine learning platform.

Concerning the manipulator, a four degrees of freedom (DOF) would be effective to execute Schönflies motions. A microcontroller was chosen as basis for the automation system. Task specification and motion planning are defined by the programs executing on the SoC board previously cited, which also works as the cell controller.

2.1 Robotic manipulator

The manipulator will be designed and built as part of the project. To do so, a PRRR serial kinematic chain was chosen (or P3R), for simplicity in modelling, design, fabrication and programming. Figure 2 presents this manipulator kinematic model. It is observed that it is similar to the classical SCARA configuration (Siciliano *et al.*, 2009), regarding the end effector posture.

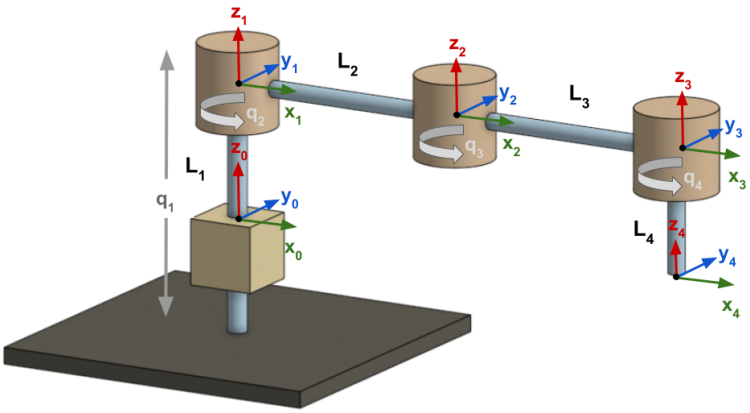

Figure 2. Kinematic chain of the P3R serial manipulator.

The robot is driven by four stepper motors. The translational motion of the first joint is generated by a lead screw that is connected to a 7 kgf.cm stepper motor. The rotational motion of joints 2 and 3 are transmitted by a timing belt/pulley system, designed to position the 4 kgf.cm stepper motors in the main body, where the first joint translates. Finally, the fourth joint is driven by a 0.35kgf.cm stepper motor directly coupled to its axis. DRV8255 drivers are used to power and command the stepper motors motion.

The manipulator control unit is composed by these drivers and an ESP32 32-bit dual core microcontroller (Espressif Systems, 2021). This MCU (microcontroller unit) also commands the end effector tool (a solenoid, for the initial case study) and monitors the state of endstop sensors positioned in the joint limits. The firmware of this MCU is responsible to calculate the steps to move the joints to the positions received from an external motion planning software and to synchronize their motion according to their velocities. In this initial version of the manipulator, an open-loop control approach was chosen, as it would be simpler to implement and to reduce costs, relying on the stepper motors precision to accurately position the joints.

Regarding the mechanical/structural parts of this manipulator, most of them will be manufactured using 3D printing from its CAD design. Other metal parts, such as rods, lead screw and bearings were found in specialized stores, as also the time belts and pulleys.

2.2 Object Localization/Identification Subsystem

This project was preceded by a previous study concerning the use of cameras and computer vision as sensor for mobile robots. In parallel, another study was made about using machine learning open source frameworks in low power embeddable computer hardware. The object localization/identification subsystem is the result of merging those studies, integrating their software and using the same hardware platform, since it produced effective results. For the initial case study, some simplifications in the computer vision processing were assumed, such as not considering the parallax effects in the object identification/positioning process and the distortion caused by the image acquisition hardware, as it could be reduced by a proper camera calibration process (Kutila *et al.*, 2001).

Initial tests used a Raspberry Pi Zero W (RPF, 2020b) as a image acquisition and processing unit – the same setup used for the computer vision implementations. However, the hardware had low performance executing machine learning frameworks tests, so it was replaced by a Raspberry Pi 3 Model B+ (RPF, 2020a). Its quad-core 1.4GHz SoC showed adequate to support the required software.

A Raspberry Pi Camera V2 module was adopted as the image capture device (RPF, 2021). It has a 8 MPixel sensor, capable of capture still images or video in high definition. This module has a dedicated serial interface with Rasperry Pi board which can achieves 1Gb/s transfer rates. Also, its driver uses the graphical processing unit (GPU) presented in the Raspberry Pi SoC, resulting in a efficient imaging processing unit. This camera was already in use for the previous studies, being a known and reliable hardware choice.

To reduce costs and ease software integration, this board was considered to execute the robotic cell controller software and the manipulator task/motion planner. This software interfaces with the MCU firmware by a serial communication protocol.

2.3 Software aspects

Python was chosen as the main development language/platform (PSF, 2021), not only because the development team had experience with the language, but also due to its wide availability of numerical/scientific APIs, large knowledge

base freely available and being an open source initiative, with an active user community (Linge and Langtangen, 2020). Regarding the MCU, Micropython was chosen as main language (Norris, 2016), since it is a lean port of Python 3 for microcontrollers, standardizing the software development (George, 2018).

A high-level class diagram containing the main software components is depicted in Fig. 3. Most components are implemented in the SoC computer board (green background), while some are implemented in the MCU (blue background).

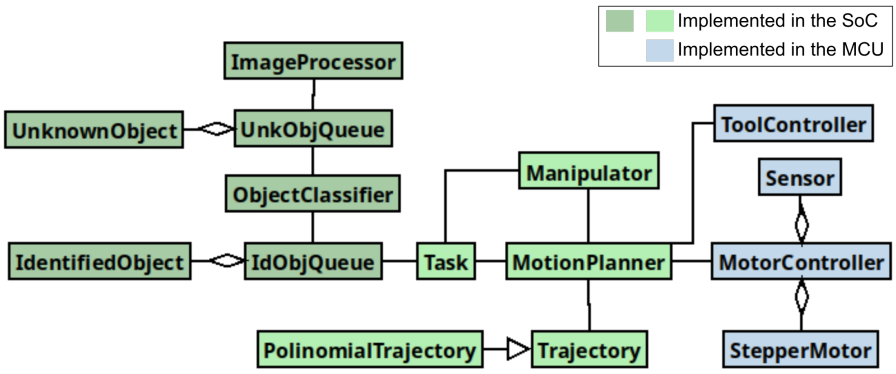

Figure 3. Robotic cell controller software main components and their relationships.

The ImageProcessor class is responsible for identifying objects in a captured image of the workspace, locating them in terms of planar coordinates (x,y) and its boundaries. Each identified object producing an instance of UnknownObject that is appended to a queue. An instance of ObjectClassifier consumes objects from this queue to classify them and, for each successfully classified object, produces an instance of IdentifiedObject, which is appended to another queue.

An instance of Task works consuming objects from the aforementioned queue, to obtain the operational space coordinates for the end effector. It uses an instance of Manipulator to determine the desired posture for the arm, solving the inverse kinematics and obtaining the joint angles. This joint space coordinates are fed to an MotionPlanner instance, which generates a trajectory for each joint.

The MotorController class has to determine the steps to be send to driver in order to move the stepper motors in a synchronized way and in the proper velocities. It also monitors joint limit sensors to avoid motions beyond these limits and possible damages to the arm. The trajectory could include commands to the end effector tool, which are executed by the ToolController class.

The task is initiated by the user. The start point is to move the manipulator to a home position, where it does not block the view of the workspace. Afterwards, the image processing stage locates the objects, while the object classifier identify them. After the classification stage, the task is planned and executed by the manipulator, stopping by user command or when the queue of identified objects is empty. Figure 4 presents the expected execution flow in an activity diagram.

3. CONSTRUCTION OF A PROTOTYPE

This project has several aspects to be addressed to build a prototype, concerning physical elements, such as the embedded electronics, mechanical and structural parts, and also software aspects, which could be subdivided between the object localization/classification and the robot controller subsystems. The robot controller can still be subdivided between the high-level software, running in the SoC, and the low-level software, running in the MCU. These parts are further discussed in the following.

3.1 Robot structural and mechanical structure

Regarding the physical robot, an initial project decision was to build it entirely. This would allow to have complete knowledge of the manipulator and full access to its components. To do so, an extensive research of similar open source projects was made, which lead to the choice of a P3R geometry. Also, it inspired the development of the CAD design model.

In the present stage, this model is under reevaluation/refactoring, since the end effector would not achieve the desired workspace horizontal plane in the first version. Also, some adjustments needed to be made to better mount the motors, pulleys and bearings.

3.2 Building the robot automation

The manipulator automation/electronics is centered around the ESP32 MCU. This microcontroller generates the necessary pulses to the drivers in order to move the stepper motors according to positions sent by the MotionPlanner class

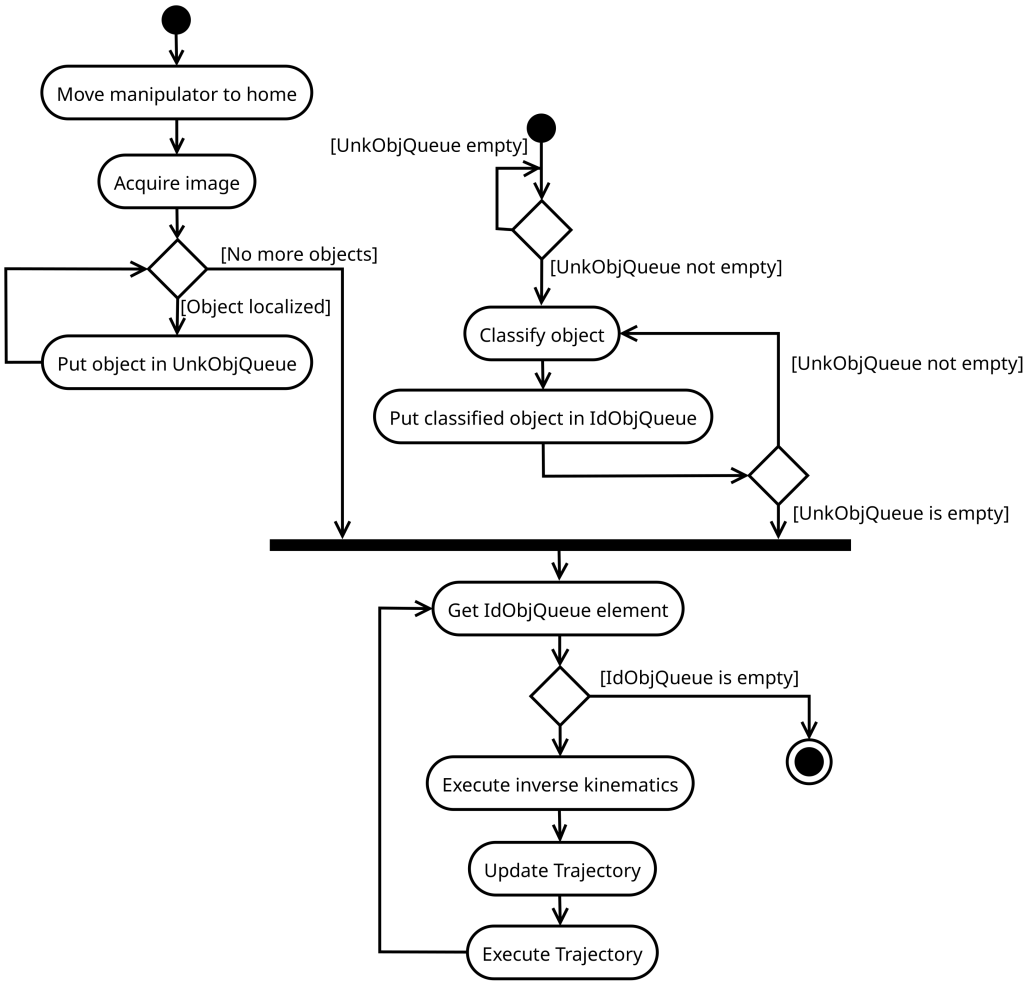

Figure 4. Controller software activity diagram.

by a serial communication interface. As sensors, there are magnetic endstops installed in the first three joint limits, and a reference endstop installed on the fourth joint (which connects the end effector to the robot). There is a output dedicated to the end effector tool, initially activating/deactivating it. However, it is possible to implement a serial interface to allow communication with an microcontrolled tool. Figure 5 illustrates this electronics in a simplified diagram.

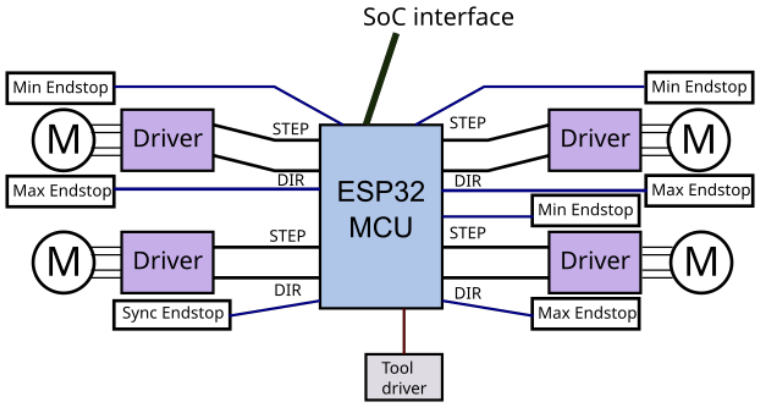

Figure 5. Manipulator control unit.

The firmware implemented the corresponding classes shown in Fig. 3. This modular approach enables easy software maintenance and extensibility. The use of Micropython has the balance between rapid development time and full access to the hardware resources as an advantage.

3.3 Development of a task management software

The Task class connects the two clearly defined parts of this project: the computer vision/machine learning subsystem to the robot controller. It receives operational position points from the first, and must generate trajectories to be followed to the last.

To do so, it is considered that each object received by the Task instance defines a point-to-point path between the end effector current position. The operational space coordinates are processed by the inverse kinematics algorithm implemented in the Manipulator class to generate the corresponding joint variables. The MotionPlanner class uses an instance of the Trajectory class to implement a joint space trajectory to be sent to the MotorController and ToolController classes (implemented as part of the ESP32 firmware).

The Denavit-Hartenberg notation was used to describe the robot kinematics (Rocha *et al.*, 2011). Its parameters are presented in Tab. 1.

Table 1. Devavit-Hartenberg parameters for the P3R manipulator

Joint	a_i	α_i	d_i	θ_i
1	0	0	q_1	0
2	l_2	0	0	q_2
3	l_3	0	0	q_3
4	0	0	$-l_4$	q_4

For this particular kinematic chain, it was considered that the geometric approach was the simplest (dos Santos *et al.*, 2006). The z end effector coordinate is defined only by the q_1 prismatic joint variable. The (x,y) coordinates depend of the q_2 and q_3 rotational joints variables. Finally, the end effector orientation can be defined by the q_4 rotational joint variable. This leads to Eq. (1), (2) and (3). The end effector orientation was not considered for the case study.

$$q_1 = z + l_4 \tag{1}$$

$$q_2 = \arctan\left(\frac{(l_1 + l_2\cos q_3)y - l_2 \sin q_3 x}{(l_1 + l_2\cos q_3)x + l_2 \sin q_3 y}\right) \tag{2}$$

$$q_3 = \pm \arccos\left(\frac{x^2 + y^2 - l_2^2 - l_1^2}{2 \times l_1 \times l_2}\right) \tag{3}$$

To provide a smooth trajectory, the Trajectory class was extended to a PolinomialTrajectory, which implements a 5^th order polinomial generator (Siciliano *et al.*, 2009). The ith joint coordinate q_i is expressed as in Eq. (4), where a_j is a polinomial coefficient and t is an instant of the total trajectory.

$$q(t) = a_5 t^5 + a_4 t^4 + a_3 t^3 + a_2 t^2 + a_1 t + a_0 \tag{4}$$

Considering a point-to-point path, it can be assumed that velocities and acceleration in initial and final instants are equal to zero, and that the initial instant is zero. Equation 5 is derived in the literature (Siciliano *et al.*, 2009). As t_f, q_i and q_f are known values (final instant, initial position and final position, respectively), the polinomial coefficients for each joint variable can be determined.

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix}^{-1} \begin{bmatrix} q(t_{in}) \\ 0.0 \\ 0.0 \\ q(t_f) \\ 0.0 \\ 0.0 \end{bmatrix} \tag{5}$$

The initial values for the joint variables correspond to the current manipulator state, while the final ones are provided by the inverse kinematic algorithm, from the next object position processed by the Task to the Motion Planner. For each path, the PolinomialTrajectory instance is updated to start a new motion for the manipulator.

3.4 Implementing the object localization/classification subsystem

As it can be observed in Fig. 3, there are two main software components: the ImageProcessor and the ObjectClassifier classes.

The ImageProcessor class has to capture workspace stills and analyze them, to identify objects and extract the necessary features to next step of classification. It also has to determine the position of the object as a (x,y) coordinate. To implement this class, Open Computer Vision (OpenCV) is used (OpenCV, 2021). It is an multiplatform open source library with several resources regarding image processing, computer vision and machine learning with interfaces for C++, Python, Java and Matlab (Howse, 2013).

To identify an object in the workspace, it is necessary to use a segmentation process in the image, partitioning it in regions according to specific features, as color or brightness, for instace. Afterwards, edge detection techniques are used to refine the segment definition, leading to contour definition. These processing methods are easily available in OpenCV and were used in this class. To determine the positions of the objects, the centroid method is used, calculated from the surrounding rectangular area of the object (Marengoni and Stringhini, 2009).

The ObjectClassifier class works with the features provided by the ImageProcessor. Different algorithms and APIs were evaluated, as discussed in Setion 4. The Convolutional Neural Network (CNN) method was adopted (Desai and Shah, 2020), implemented using the Keras API, which was based on the Tensorflow library (Keras SIG, 2021).

In order to work properly, the CNN must undergo a training process. For this project, it was used a supervised learning technique, consisting of providing a training set composed by pictures of the types of objects to be classified, along with their labels. The ObjectClassifier class implemented this training feature, so it can be trained for different types of objects. The trained models are saved in the *tflite* format, which allows to create different previously trained models which could be loaded to the ObjectClassifier to execute several classification tasks. Other interesting characteristic of this feature is that the training can be executed in different (even more powerful) computers, which do not require the robotic cell to stop for training a new classification set. The Tensorflow Lite is also suitable for low power devices, as smartphones, tablets and even high end microprocessors (TensorFlow, 2021).

The final result of the ObjectClassifier is an object that contains the position of the object and its highest probable type along with its probability, which can be used to decide if the identification is reliable or the object should be discarded.

4. INITIAL RESULTS

The early results obtained so far are detailed in this section. It is mainly focused on software implementations, since the physical manipulator is still under construction.

4.1 Manipulator controller software

The Manipulator class was implemented, representing the P3R kinematic model. Besides maintaining the robot state variables, the class has methods to calculate direct and inverse kinematics. The component was tested in simulations, where it was verified that inverse and direct kinematics algorithms produce the right results.

The Trajectory class and its derived class – PolinomialTrajectory – were also implemented and tested, and the results were accurate.

Initial tests integrating the high-level robot controller (components implemented in the SoC) indicate that the task resolution is functional. However, it requires further testing.

Regarding the MCU software components, the MotorController class calculates the right amount of steps to drive the stepper motors to the desired positions, even synchronizing the motion of all actuators to provide a continuous path to the end effector. This initial tests were made in debug mode, were the pulses were displayed in a textual terminal. The next tests will include integration with the StepperMotor instances, to actually command the drivers. The Sensor and ToolController are implemented and fully functional.

4.2 Evaluation of machine learning algorithm

As it was desired to ease implementation of machine learning algorithms, a design choice was made, to use libraries that provide the most used ones for image classification available for Python. So, Scikit-learn (Pedregosa *et al.*, 2011) and Keras (Keras SIG, 2021) were selected, based on the number of existing work using them and the available documentation. Scikit-learn provided random forest (Akar and Güngör, 2012) and multi-layer perceptron (MLP) (Desai and Shah, 2020) implementations (scikit learn, 2021a,b). The Tensorflow based Keras API provided the CNN implementation (Keras SIG, 2021). These three algorithms were submitted to the same test setup, in order to select the best suited for the proposed task.

The test set was comprised of 180 pictures of randomly oriented nuts and bolts, as it was the initial case study. Each picture has 480x480 pixel in RGB format. The set had 50% of bolt and 50% of nut pictures.

After training, a validation set (variations of the initial set) was used to verify the accuracy of each algorithm. The results summarized in Fig. 6 were compared, and despite possible differences in implementation and the randomness of the algorithms initialization, it was considered that the CNN algorithm had the highest accuracy in classification, and it was chosen to implement the ObjectClassifier class.

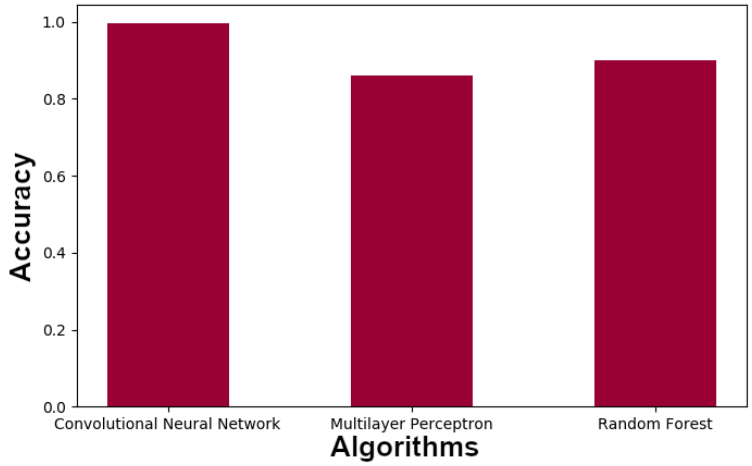

Figure 6. Accuracy of different classification algorithms for the case study.

4.3 Image acquisition and processing

The ImageProcessor class used OpenCV for accessing the camera, execute image segmentation and determine positions. To test the implementation and its connection to the ObjectClassifier instance, a batch of 30 image capture/processing tasks was executed. In each case, between 0 and 5 objects were randomly positioned in the workspace, and the ImageProcessor should determine the positions and segment the image to feed the previously trained ObjectClassifier. In this initial batch, all objects were correctly located and classified. Since the complete cell is not yet mounted, this test is considered only to verify the correctness of the implementations and the adequate connection between the two important software components.

5. FINAL REMARKS

This paper presented the early results of a project whose objective is to create a robotic cell for research and study of computer vision, machine learning and their integration in robotics. This is a feature increasingly required for applications that must be flexible in manufacturing processes in industry, among others. As a work in progress, there is still several steps to conclude the initial prototype. However, the results obtained so far are relevant, especially considering that this project is a final term course for a technical course, and the students do not formally study the main research fields covered in this work.

So far, the software implementation is nearly completion, and the automation system is already tested. As soon the mechanical/structural parts of the manipulator are completed, initial integration tests and calibration will be made, and this results will be updated in future publications.

Once completed, this project can lead to several future developments exploring robotics, computer vision and other related fields. For instance, the manipulator posture could be estimated from a video feed, providing data for implementing a closed-loop control (at least to some extent). Other sensors could be used to implement this closed-loop control, such as inertial measurement units. Other machine learning techniques can be easily implemented, due to the software modularity. Finally, this robotic cell can be a valuable learning resource for automation and robotics in different levels.

As soon the prototype is concluded and validated, the full project will be available in an open source repository, to be replicated and extended. This would be a complementary contribution of this work.

6. ACKNOWLEDGEMENTS

The authors thank the partial support granted by the Federal Institute of Education, Science and Technology of Rio Grande do Sul.

7. REFERENCES

Akar, Ö. and Güngör, O., 2012. "Classification of multispectral images using random forest algorithm". *Journal of Geodesy and Geoinformation*, Vol. 1, No. 2, pp. 105–112.

Desai, M. and Shah, M., 2020. "An anatomization on breast cancer detection and diagnosis employing multi-layer perceptron neural network (mlp) and convolutional neural network (cnn)". *Clinical eHealth*.

dos Santos, F.M., Watanabe, É.T. and Carrara, V., 2006. "Estudo da cinemática inversa aplicada num braço robótico".

Anais do 12º Encontro de Iniciação Científica e Pós-Graduação do ITA–XII ENCITA.

El-Shamouty, M., Kleeberger, K., Lämmle, A. and Huber, M., 2019. "Simulation-driven machine learning for robotics and automation:". *tm - Technisches Messen*, Vol. 86, No. 11, pp. 673–684. doi:doi:10.1515/teme-2019-0072. URL https://doi.org/10.1515/teme-2019-0072.

Espressif Systems, 2021. *ESP32 Datasheet*. Espressif Systems, Shangai.

George, D., 2018. "Micropython - python for microcontrollers". George Robotics, https://micropython.org/. Accessed 24 may 2021.

Howse, J., 2013. *Opencv Computer Vision with Python*. Packt Publishing. ISBN 1782163921.

Keras SIG, 2021. "Keras: The python deep learning api". Keras, https://keras.io/. Accessed 07 may 2021.

Kutila, M., Korpinen, J. and Viitanen, J., 2001. "Camera calibration in machine automation". In E. Arai, T. Arai and M. Takano, eds., *Human Friendly Mechatronics*, Elsevier Science, Amsterdam, pp. 211–216. ISBN 978-0-444-50649-8. doi:https://doi.org/10.1016/B978-044450649-8/50036-X. URL https://www.sciencedirect.com/science/article/pii/B978044450649850036X.

Linge, S. and Langtangen, H.P., 2020. *Programming for Computations - Python*. Springer International Publishing, Cham. ISBN 978-3-030-16877-3. doi:10.1007/978-3-030-16877-3.

Marengoni, M. and Stringhini, S., 2009. "Tutorial: Introdução à visão computacional usando opencv". *Revista de Informática Teórica e Aplicada*, Vol. 16, No. 1, pp. 125–160.

Melis, M., Demontis, A., Biggio, B., Brown, G., Fumera, G. and Roli, F., 2017. "Is deep learning safe for robot vision? adversarial examples against the icub humanoid". In *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops*.

Norris, D., 2016. *Python for Microcontrollers: Getting Started with MicroPython*. McGraw-Hill Education, New York. ISBN 978-1-25-964453-5.

OpenCV, 2021. "About - opencv". OpenCV, https://opencv.org/about. Accessed 04 june 2021.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E., 2011. "Scikit-learn: Machine learning in Python". *Journal of Machine Learning Research*, Vol. 12, pp. 2825–2830.

Piccinini, P., Prati, A., Cucchiara, R., DII, D.S. and Emilia, R., 2009. "Multiple object detection for pick-and-place applications." In *MVA*. pp. 362–365.

PSF, 2021. "About python". Python Software Foundation, https://python.org/about. Accessed 05 april 2021.

Rocha, C., Tonetto, C. and Dias, A., 2011. "A comparison between the denavit–hartenberg and the screw-based methods used in kinematic modeling of robot manipulators". *Robotics and Computer-Integrated Manufacturing*, Vol. 27, No. 4, pp. 723–728.

Romano, V. and Dutra, M., 2002. "Introdução à robótica industrial". *Robótica Industrial: Aplicação na Indústria de Manufatura e de Processo, São Paulo: Edgard Blücher*, pp. 1–19.

RPF, 2020a. "Raspberry pi 3 model b+". Raspberry Pi Foundation, https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/. Accessed 14 may 2021.

RPF, 2020b. "Raspberry pi zero w". Raspberry Pi Foundation, https://www.raspberrypi.org/products/raspberry-pi-zero-w/. Accessed 14 may 2021.

RPF, 2021. "Camera module - raspberry pi documentation". Raspberry Pi Foundation, https://www.raspberrypi.org/documentation/hardware/camera.

scikit learn, 2021a. "sklearn.ensemble.randomforestclassifier". scikit-learn 0.24.2 Documentation, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. Accessed 14 may 2021.

scikit learn, 2021b. "sklearn.neural_network.mlpclassifier". scikit-learn 0.24.2 Documentation, https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html. Accessed 14 may 2021.

Siciliano, B., Sciavicco, L., Villani, L. and Oriolo, G., 2009. *Robotics: Modelling, Planning and Control*. Advanced Textbooks in Control and Signal Processing. Springer, London.

Sünderhauf, N., Brock, O., Scheirer, W., Hadsell, R., Fox, D., Leitner, J., Upcroft, B., Abbeel, P., Burgard, W., Milford, M. and Corke, P., 2018. "The limits and potentials of deep learning for robotics". *The International Journal of Robotics Research*, Vol. 37, No. 4-5, pp. 405–420. doi:10.1177/0278364918770733. URL https://doi.org/10.1177/0278364918770733.

TensorFlow, 2021. "Guia do tensorflow lite". Tensorflow, https://www.tensorflow.org/lite/guide. Accessed 27 may 2021.

Wang, Z., Li, H. and Zhang, X., 2019. "Construction waste recycling robot for nails and screws: Computer vision technology and neural network approach". *Automation in Construction*, Vol. 97, pp. 220–228. ISSN 0926-5805. doi:https://doi.org/10.1016/j.autcon.2018.11.009. URL https://www.sciencedirect.com/science/article/pii/S0926580518302218.

8. RESPONSIBILITY NOTICE

The authors are solely responsible for the printed material included in this paper.