Proceedings of the ASME 2021 &
International Mechanical Engineering Congress and Exposition
IMECE 2021
November 1-5, 2021, Virtual, Online

IMECE2021-71479

FRAMEWORK FOR AUTOMATED ROBOTIC ARM MANIPULATION IN VARIABLE INDUSTRIAL ENVIRONMENTS

Anvay A. Pradhan
Controls, Automation, and Robotics Laboratory
Department of Mechanical Engineering
University of Iowa
Iowa City, Iowa 52240
Email: anvay-pradhan@uiowa.edu

Will C. Martin
Controls, Automation, and Robotics Laboratory
Department of Mechanical Engineering
University of Iowa
Iowa City, Iowa 52240
Email: will-martin-1@uiowa.edu

Juliana Danesi Ruiz
Controls, Automation, and Robotics Laboratory
Department of Mechanical Engineering
University of Iowa
Iowa City, Iowa 52240
Email: juliana-danesiruiz@uiowa.edu

Phillip E. Deierling
Controls, Automation, and Robotics Laboratory
Department of Mechanical Engineering
University of Iowa
Iowa City, Iowa 52240
Email: phillip-deierling@uiowa.edu

ABSTRACT

In this paper, we present a generalized, holistic method for automated robotic arm handling of manufactured components in an industrial setting using computer vision. In particular, we address scenarios in which a high volume of manufactured parts are moving along a conveyor belt at random locations and orientations with multiple robotic arms available for manipulation. We also present specific, tested solutions to all stages of the framework as well as some alternative methods based on the literature review. The framework consists of three stages: (1) visual data capture, (2) data interpretation, and (3) command generation and output to robotic arms. In the visual data capture stage, a multi-component computer vision system takes in a live camera feed and exports it to an external processor. In the data interpretation stage, this video feed is interpreted using tools like 3D point clouds and object detection/tracking models to provide useful information such as object number, location, velocity, and orientation. Lastly, the command generation and output to the robotic arms stage takes the information acquired from the analysis in the data interpretation stage and turns it into instructions for robot control. While a full-scale, cohesive system has yet to be tested, our solutions to each stage show the feasibility of implementing such a system in an industrial setting.

1. INTRODUCTION

Modern, large-scale manufacturing lines often rely on robots to automate repetitive, predictable, and sometimes dangerous tasks [1]. [1] highlights examples of robots creating safer and more dependable environments for monotonous tasks like painting and welding in automobile manufacturing. While these robots are fast and efficient, the tasks they can currently accomplish are limited. These robots simply repeat tasks in the same fashion for hours on end. In such systems, variability of the task has been far reduced or eliminated to allow for robot handling. However, many tasks involve a level of variability that existing robots are not equipped to handle, usually requiring human manipulation [2]. Niola et al. in [2] verifies this argument by noting the simplifications required to implement a control algorithm for existing robotic handling systems. In addition to being a safety risk, human manipulation may not produce the degree of accuracy required for desired quality control due to the repetition and fatigue resulting from prolonged work shifts and large production demands as [3] notes. Having a streamlined robotic system capable of dealing with and adapting to variability in a system would expedite the manufacturing process, increase efficiency, and ensure worker safety.

Variability and adaptability in a robotic system allow for a less monotonous application. This is desirable as robot applications for individual industries can vary greatly. As an example, consider arbitrary factories A and B, both utilizing a robotic arm. Factory A has parts which are picked by the robot at a predefined location on a stationary surface, while factory B has parts at random locations along a moving conveyor. While factory B will obviously require a more complex system to retrieve parts, similar components of a single method can be implemented in both factories A and B to streamline the process. This includes the camera, detection algorithm, and robot communication used. Thus, when designing a robotic system, it is important to take into account some adaptability and transferability to other applications. In this paper, we focus on designing a model for a conveyor system, but our methods are adaptable to other industrial environments as well.

In many manufacturing lines, a key component is the conveyor belt [4]. A conveyor belt can carry a set of parts with varying quantity, location, and geometry. In order for a robot to be introduced into current systems, pre-manipulation is required [4]. Pre-manipulation refers to the adjustment of parts moving along a manufacturing process, usually on a conveyor belt. Such adjustment aligns parts on the conveyor making it possible to implement robotic systems. While pre-manipulation using rollers or guides eliminate inherent variations in transportation, they often induce additional costs, introduce errors, and reduce production rates [4].

Thus, this study was motivated by a desire to utilize robotic arms in industrial systems involving conveyor belts without the need for human manipulation or pre-manipulation. Because of this constraint, the problem becomes considerably more difficult, requiring detection and tracking prior to manipulation of parts along the conveyor. Additional considerations for this study included versatility such that the framework can be introduced into environments with or without pre-existing robotic arm processes.

In this study, we present a framework by which a stationary robotic arm can detect, locate, and retrieve a part at any location along a moving conveyor. We also present tested solutions to the various stages of the framework and discuss how each stage works jointly for a streamlined process. While these solutions highlight the hardware and software utilized, the main focus of this paper is on the structuring and combining of previous tools and methods for use in a useful, unique application.

2. RELATED WORK

Object detection and grasping have been major areas of computer vision research for many years. Within the past 10 years, researchers have begun to implement machine learning models to improve the accuracy and speed that an object can be detected, and further, grasped.

The goal of most object detection models is to locate and classify objects within an image. The output of these models typically consists of a large number of rectangular bounding boxes with corresponding classification predictions. Most modern object detection models can be categorized into two classes: region proposal-based frameworks and regression/classification-based frameworks [5].

Region proposal-based frameworks are built on multi-step models that choose specific regions of interest in an image to focus on. A convolutional neural network (CNN) is then used to find object bounding boxes and classification predictions. Popular examples of this type of model include R-CNN [6], SPP-Net [7], Fast R-CNN [8], and Faster R-CNN [9]. These models can produce very accurate results but are typically too slow for real-time applications.

Regression/classification-based frameworks were designed to operate much quicker than region proposal-based frameworks and can operate in a single step. A prominent model of this type is You Only Look Once (YOLO) [10] which operates with a fixed anchor box grid to predict object locations. YOLO detection operates at high speeds, but does not handle small or clustered objects well. The Single Shot Detector (SSD) [11] model was designed to improve on YOLO by adaptively placing anchor boxes with varying sizes and aspect ratios. A variant of SSD called SSDLite [12] has been designed for use on devices with low computing speeds. Popular backbones for SSDLite, such as MobileNetV3 [13], aid in improving the model's performance further. This project utilized the Mobiledet [14] backbone which is designed specifically to increase the performance of the SSDLite model on mobile accelerators such as a Coral Edge Tensor Processing Unit (TPU).

Many different strategies have been explored to predict grasps for a two-finger robot end-effector using machine learning models. One of the most common methods involves generating grasping points in a 2D plane from an image of a previously unseen object. The model design presented in [15], built on a Generative Grasping Convolutional Neural Network (GG-CNN) is an example of this type of model. This model takes a 2D depth image as an input and returns the grasp quality, width, and angle at each pixel in the image. The optimal grasp can then easily be extracted from the output. This simple, fully convolutional model architecture has only $\sim$62k parameters making it the ideal candidate to use on mobile accelerators like the one used in this research.

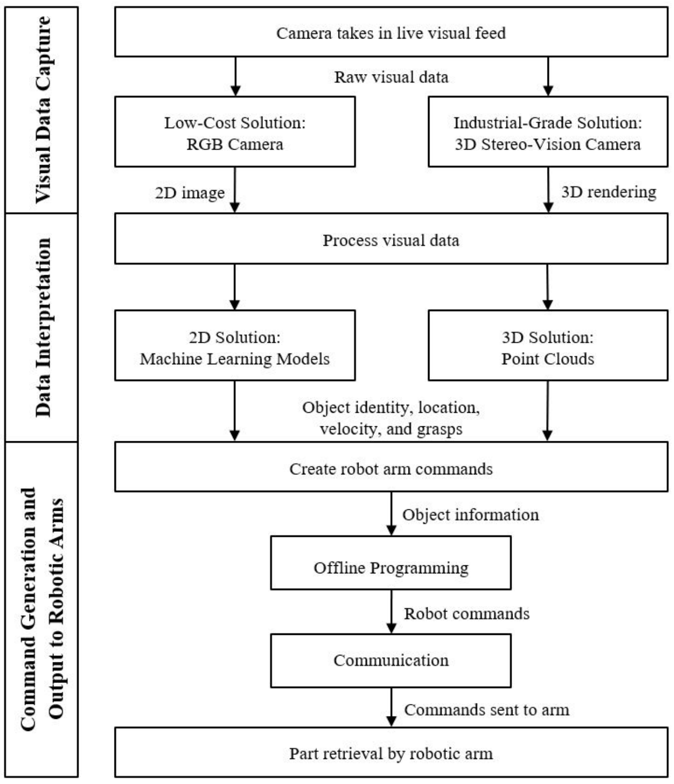
There is currently a high demand for industrial vision systems which can accurately control a robot arm utilizing methods similar to those discussed above [16]. Existing products like Roboception cameras can be slow or expensive, making them less than ideal for use in an industrial application. One of the most commonly used brands for robotic arms in manufacturing is KUKA. The communication between an external vision system and a KUKA robot is available through KUKA Ethernet KRL. The connection is established using an XML file configuration. [17] explains that even though the communication is relatively slow, Ethernet KRL is effective and accurate.

3. METHODS

We have developed a simple framework to enable efficient interaction between robotic arms and objects moving along a conveyor system. The framework is centered around three major steps: (1) visual data capture, (2) data interpretation, and (3) command generation and output to robotic arms. Each stage is broadly defined and can be achieved using a number of tools/methods. Figure 1 depicts the data flow of this framework graphically.

3.1 Visual Data Capture

In the visual data capture stage, a multi-component computer vision system takes in a live camera feed and exports it to an external processor. The type of camera utilized dramatically affects how data interpretation in the next stage takes place. In particular, we tested two vastly different camera systems: a 3D, color, stereo-vision, Roboception camera versus a stationary, low-cost, RGB, Raspberry Pi 4 camera. The low-cost solution was motivated by initial testing done using the Roboception cameras which had high cost, were difficult in implementation, and lacked of compatibility with existing systems.



Visual Data Capture
Data Interpretation
Command Generation and Output to Robotic Arms
Camera takes in live visual feed
Raw visual data
Low-Cost Solution: RGB Camera
Industrial-Grade Solution: 3D Stereo-Vision Camera
2D image
3D rendering
Process visual data
2D Solution: Machine Learning Models
3D Solution: Point Clouds
Object identity, location, velocity, and grasps
Create robot arm commands
Object information
Offline Programming
Robot commands
Communication
Commands sent to arm
Part retrieval by robotic arm

FIGURE 1: FLOWCHART OF DATA IN FRAMEWORK FOR AUTOMATED ROBOTIC ARM MANIPULATION

3.1.1 3D-Roboception

For the industrial version of visual data capture, a stationary Roboception camera, specifically the rc_viscard 160, was used. This camera uses stereo vision for depth-sensing and color vision for area distinction. When initially implemented, Roboception's Graphical User Interface (GUI) rendered three camera visuals: a standard image, a depth image, and a correspondences image. However, this GUI simply displayed the visuals, offering no method of utilizing this visual data. An open source method was used to interpret the data. First, a proprietary library was constructed that consisted of various open source binaries and packages from Roboception as well as Point Cloud Library (PCL) GitHub pages. Next, the camera visual was activated via command prompt and calibrated so depth readings could be translated into actual distances from the camera. A 2D grid was then overlaid on the standard image visual at equal spacings on the image area. These spacings could be adjusted based upon desired resolution and therefore accuracy, but in general, the spacing was chosen such that the number of points was less than the number of pixels.

Using a depth image like the one seen in Figure 2, each point on the 2D grid was assigned a height value which could be adjusted based on the distance of the camera from the work surface. Data was sampled using the 2D spacing instead of the full image area to speed up processing and obtain a continuously-updating, live 3D point cloud. While sacrificing some resolution, a 2D grid with depth visual overlaid proved to be much quicker at displaying the 3D point cloud than rendering the full image space.

3.1.2 2D-Raspberry Pi

The low-cost visual data capture solution was designed around a Raspberry Pi 4 Model B and a Raspberry Pi Camera Module v1.3. The Raspberry Pi used for this project had a quad-core ARM Cortex-A72 processor with 4 GB RAM. The camera used a OmniVision OV5647 sensor and was configured to capture video at 640×480p with a rolling shutter.

The camera was mounted at a fixed location overlooking the work area. This was done so that the spatial location of detected objects could easily be determined in relation to the robotic end effector.

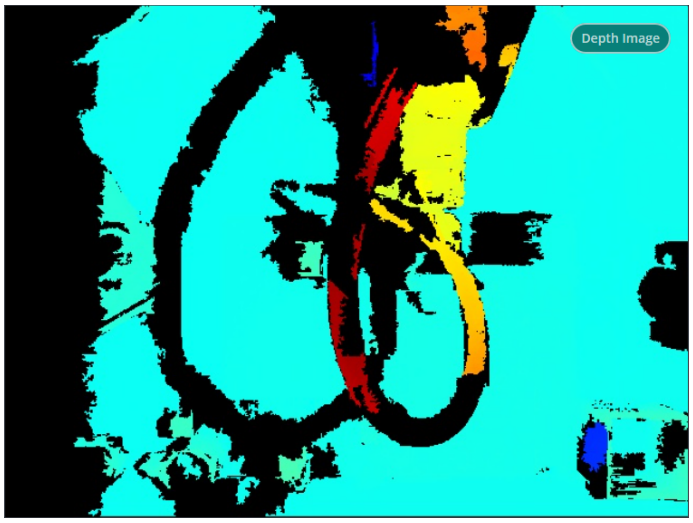


FIGURE 2: ROBOCEPTION DEPTH IMAGE OF KUKA KORE KR6 R700 ROBOT HOUSING. BLUE INDICATES OBJECTS FURTHER AWAY FROM THE CAMERA WHILE RED INDICATES OBJECTS CLOSER TO THE CAMERA.

3.2 Data Interpretation

In the data interpretation stage, the video feed was interpreted using tools like 3D point clouds and object detection/tracking. These models provide useful object information such as number, location, velocity, and orientation.

3.2.1 3D-Point Clouds Point clouds are virtual recreations of real-life, 3D objects [18]. There exist two methods to acquire a point cloud for extracting object information: rigid camera mount or dynamic camera mount. Because of the manner in which the coordinate frames vary for each, rendering of the point cloud, and more importantly the calibration of the point cloud, can vary. However, rendering a point cloud alone does not give useful information, but it does provide a virtual model of a live 3D space to which different processing techniques can be applied. For instance, because of their depth-sensing capability, point cloud models can be designed to detect changes in depth alone and parse those changes in its image space as movements of one or more objects.

In addition, point clouds can be coupled with machine learning models to accurately find the true properties of an object. In 2D space, height must be estimated and is most commonly some fixed value, however point clouds enable the third dimension to be known.

3.2.2. 2D – Machine Learning Models Once a 2D video feed had been established from a Raspberry Pi, information needed to be extracted from each frame. For a robotic arm to

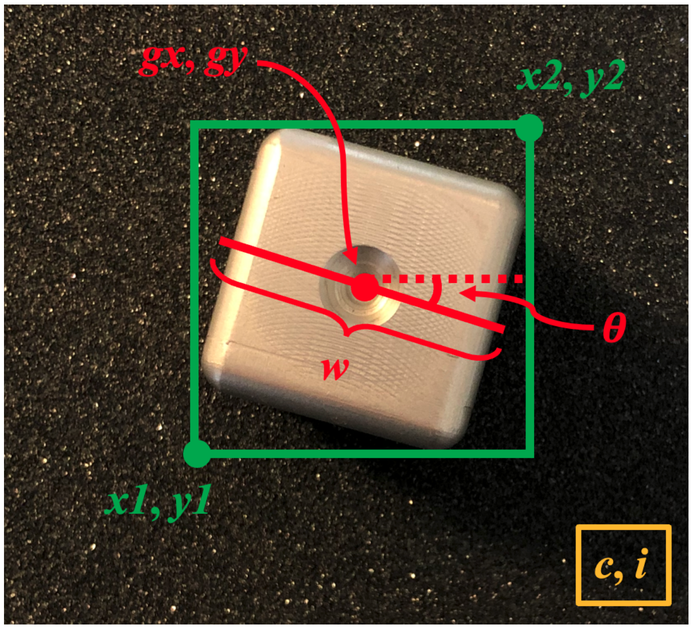


FIGURE 3: AN OBJECT IS CHARACTERIZED AS $\mathbf{o} = (c, i, \mathbf{bb}, \mathbf{g})$ WITH $\mathbf{bb} = (x1, y1, x2, y2)$ AND $\mathbf{g} = (gx, gy, \theta, w)$. THESE VARIABLES ARE SHOWN IN THE FIGURE WITH RELATION TO A SIMPLE CUBE.

grasp objects on a conveyor system, object identities, positions, and grasp locations needed to be calculated. This was achieved using trained machine learning models and a simple tracker.

Let $\mathbf{o} = (c, i, \mathbf{bb}, \mathbf{g})$ define an object as seen in Figure 3. The object is first specified by its classification, c. The classification is recorded as an integer scalar that corresponds to a part in a catalog of possible components. The object is assigned a unique identification number, *i*. The identification number must be maintained whenever the object is seen by the camera. This allows multiple objects of the same classification to be individually identified for tracking and grasping purposes.

The spatial location of an object is defined by a bounding box, $\mathbf{bb} = (x1, y1, x2, y2)$. The bounding box specifies the coordinates of the lower-left and upper-right corners of a rectangle containing the object. The box is aligned with the image frame and lays in a plane parallel to the workspace.

Finally, a grasp, $\mathbf{g} = (gx, gy, \theta, w)$ is defined. The grasp consists of gx and gy specifying the center of the grasp in the same plane as the bounding box, θ describing the rotation of the grasp, and w representing the width of the grasp. The robot Tool Point Center (TPC) travels perpendicular to the plane the grasp is defined in when attempting to pick up objects.

Similar to [15], **bb** and **g** will first both be measured in a coordinate system defined by the camera and will need to be transformed into the world coordinate system occupied by the robot. This can be done by applying a transformation based on the camera calibration.

An image of dimension [640, 480, 3] was captured from the Raspberry Pi RGB camera feed, cropped to square, and scaled to [320, 320, 3]. Every n frames, a Mobiledet+SSDLite [14] object detection model was used to identify object bounding boxes, **bb**, and their corresponding classifications, c. A simple center point comparison algorithm was run on all new bounding boxes to match them to the bounding boxes in the previous frame. The algorithm either creates a new unique identification number, i, for objects that are just appearing in the frame or maintains the identification number for objects that were previously in frame. Objects leaving the frame and re-entering was considered an insignificant issue as the conveyor was considered to only move in one direction.

Next, a GG-CNN model designed by Morrison et al. [15] was used to find suitable grasps, **g**, on the objects in frame. For each object, a modified frame was created where the area of the image outside of the object's bounding box was blurred. Due to the blurring, when each modified frame was input to the GG-CNN only grasps inside of the bounding box, and thus on the specified object, were output. This allowed for the production of grasps specific to individual objects even when multiple objects were in the camera's view.

During frame numbers not divisible by n, an object tracker [19] was run to maintain the bounding box and corresponding grasp for each object already detected.

The general algorithm used to identify m objects is summarized in Figure 4. Sample outputs are visualized on input images of a test object in Figure 5.

Speed was a major concern while designing this system. For the system to be successful, each frame needed to be processed at a speed that would allow the robot to grasp objects as they passed on a conveyor moving at a manufacturing pace. The processing speed of the system was first increased by only running the computationally expensive machine learning models every n frames. Between these frames a much lower cost object tracker was run to calculate an estimated position for each object. As n was increased, the frame processing rate increased while the overall accuracy of object location prediction decreased. The optimal n to balance this tradeoff was found to be 10 ± 5 frames depending on the rate of objects passing the vision system.

Additionally, the computing speed of the two machine learning models posed a considerable issue. The unaltered models were unable to process faster than 1-2 seconds per inference on the Raspberry Pi CPU. To increase the processing speed, the models were run on a low-cost, external Coral Edge TPU. In recent years, the Coral Edge TPU has become an extremely common tool to run small machine learning, particularly vision, models for both researchers and hobbyists [20]. Both models were compiled to use only 8-bit integer numbers through all layers, rather than the original 32-bit floating-point numbers, to increase computation time and to allow compatibility with the TPU [21].

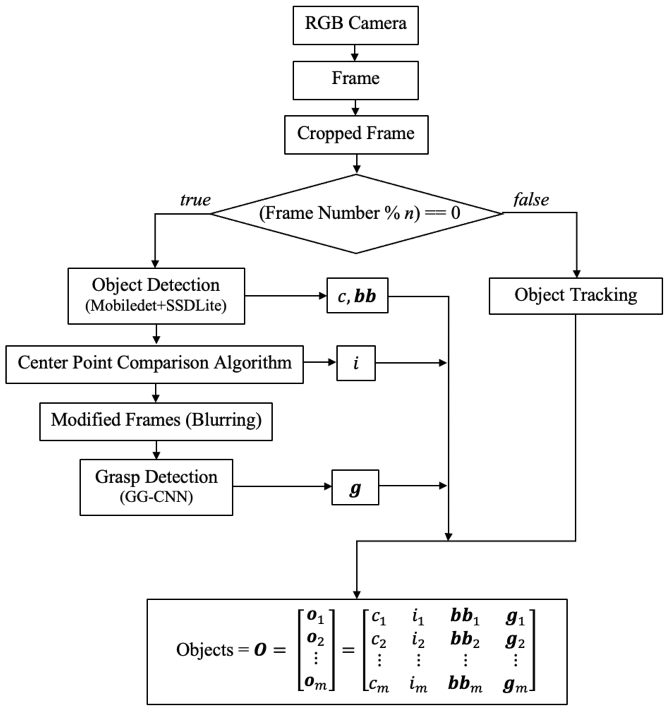

RGB Camera
Frame
Cropped Frame
true
(Frame Number % n) == 0
false
Object Detection (Mobiledet+SSDLite)
$c, \boldsymbol{bb}$
Object Tracking
Center Point Comparison Algorithm
i
Modified Frames (Blurring)
Grasp Detection (GG-CNN)
$\boldsymbol{g}$
Objects = $\boldsymbol{O}$ = $\begin{bmatrix} \boldsymbol{o}_1 \\ \boldsymbol{o}_2 \\ \vdots \\ \boldsymbol{o}_m \end{bmatrix} = \begin{bmatrix} c_1 & i_1 & \boldsymbol{bb}_1 & \boldsymbol{g}_1 \\ c_2 & i_2 & \boldsymbol{bb}_2 & \boldsymbol{g}_2 \\ \vdots & \vdots & \vdots & \vdots \\ c_m & i_m & \boldsymbol{bb}_m & \boldsymbol{g}_m \end{bmatrix}$

FIGURE 4: THE ALGORITHM USED TO IDENTIFY m OBJECTS USING THE 2D DATA CAPTURE METHOD.

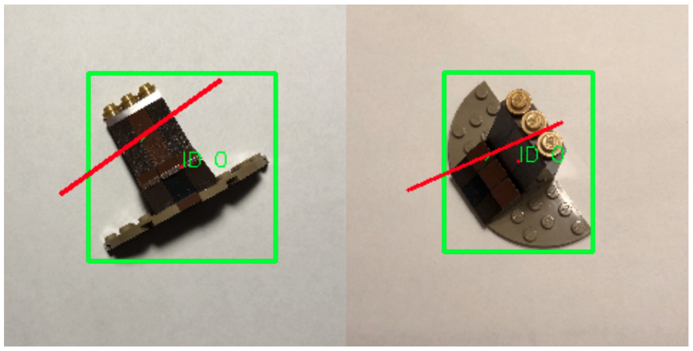


FIGURE 5: SAMPLE OUTPUTS FROM THE 2D DATA INTERPRETATION METHOD. OBJECT BOUNDING BOXES ARE SHOWN IN GREEN WITH GRASP PREDICTIONS DISPLAYED IN RED.

This decreased the models' accuracy slightly, but allowed for real-time run speeds. To compute an inference on one image, the Mobiledet+SSDLite model takes 10 ms while the GG-CNN model takes 45 ms on the specified hardware.

The original GG-CNN grasp detection model created by [15] was designed for a depth image of size $[300, 300, 1]$. Since our system uses an RGB camera without a depth measurement, the input was modified to accept an image of size $[300, 300, 3]$. After initial experimentation, there did not seem to be a considerable accuracy decrease with this modification. The model was retrained on RGB images from the Cornell Dataset [22] with random alterations to generate a greater variety of images. This procedure was identical to that used by [15]. The training procedure will be updated soon to utilize the larger and more diverse Jacquard Dataset [23].

The SSD object detection model was retrained on labeled images of test objects. Training was performed with ~ 400 images per class using images taken with varying lighting conditions, occlusion, and camera angle.

Both models were trained on an NVIDIA Tesla T4 available through the cloud-based Google Colab service.

3.3. Command Generation and Output to Robotic Arms

This stage consists of two major components: communication and offline programming. Communication takes place in real-time during robot operation. Offline programming is done beforehand to ensure commands can be created given data input from the previous stage.

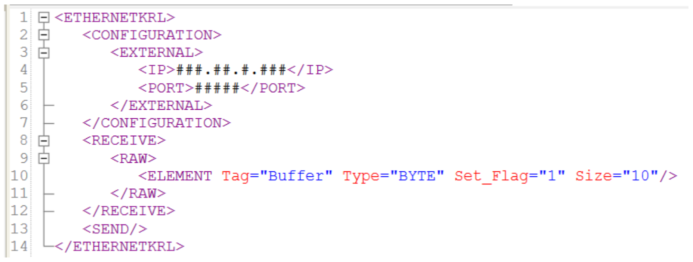
3.3.1. Communication

In order to communicate with the KRC Compact and the KUKA Sunrise Cabinet robot controllers, Ethernet KRL interface was used. As [24] explains, this interface allows the communication between a computer and a KUKA robot through a TCP/IP protocol. The Ethernet KRL connection must be configured using an XML file. Figure 6 shows an example of the XML file. It defines the external system communication as either a server or client, with the default configuration as server. Then, it uses the IP and an available port number of the external system to communicate. There are various attributes that can be defined in the XML file, such as information size. In Figure 6, the information size is set to 10 bytes.

In addition to the XML configuration file, Ethernet KRL has two other components – the application and program. Figure 7 demonstrates a example of an Source Code (SRC) program, which contains the commands to initialize and close the connection as well as facilitate any exchange of information between systems.

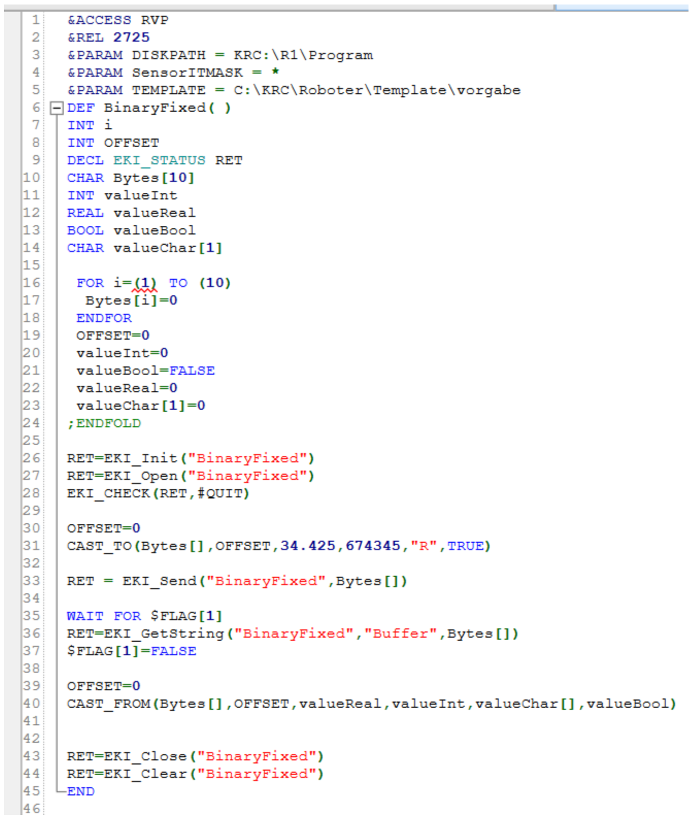
3.3.2. Offline Programming

The two robotic arms utilized in this work, the KUKA LBR iiwa and KUKA KORE, operate using distinct programming languages. While KUKA Robot Language (KRL) used on the KUKA KORE is intuitive, KUKA Sunrise.OS used on the LBR iiwa is more complex. Thus, it was necessary to research universal software that is straightforward to use. In addition, due to limitations in lab occupancy resulting from the COVID-19 pandemic, a new method was rethought to enable setup validation prior to implementation. Offline robot



```
<ETHERNETKRL>
   <CONFIGURATION>
      <EXTERNAL>
         <IP>###.##.#.###</IP>
         <PORT>#####</PORT>
      </EXTERNAL>
   </CONFIGURATION>
   <RECEIVE>
      <RAW>
         <ELEMENT Tag="Buffer" Type="BYTE" Set_Flag="1" Size="10"/>
      </RAW>
   </RECEIVE>
   <SEND/>
</ETHERNETKRL>
```

FIGURE 6: XML CONFIGURATION FILE EXAMPLE.



```
&ACCESS RVP
&REL 2725
&PARAM DISKPATH = KRC:\R1\Program
&PARAM SensorITMASK = *
&PARAM TEMPLATE = C:\KRC\Roboter\Template\vorgabe
DEF BinaryFixed( )
INT i
INT OFFSET
DECL EKI_STATUS RET
CHAR Bytes[10]
INT valueInt
REAL valueReal
BOOL valueBool
CHAR valueChar[1]

 FOR i=(1) TO (10)
  Bytes[i]=0
 ENDFOR
 OFFSET=0
 valueInt=0
 valueBool=FALSE
 valueReal=0
 valueChar[1]=0
;ENDFOLD

RET=EKI_Init("BinaryFixed")
RET=EKI_Open("BinaryFixed")
EKI_CHECK(RET,#QUIT)

OFFSET=0
CAST_TO(Bytes[],OFFSET,34.425,674345,"R",TRUE)

RET = EKI_Send("BinaryFixed",Bytes[])

WAIT FOR $FLAG[1]
RET=EKI_GetString("BinaryFixed","Buffer",Bytes[])
$FLAG[1]=FALSE

OFFSET=0
CAST_FROM(Bytes[],OFFSET,valueReal,valueInt,valueChar[],valueBool)


RET=EKI_Close("BinaryFixed")
RET=EKI_Clear("BinaryFixed")
END
```

FIGURE 7: SRC FILE EXAMPLE – ETHERNETKRL APPLICATION.

simulators allow researchers to create and run various robot applications while ensuring collision avoidance and user safety. While KUKA offers a proprietary simulation software called KUKA Sim Pro, this software is exclusive to KUKA robot applications, is expensive, and has a lack of online resources, making it unappealing for this project.

RoboDK on the other hand, is a more intuitive and affordable software that allows novice users to create various robot environments [25]. The software can be easily programmed in high-level programming languages, such as Python and MAT-

LAB. Additionally, as [26] explains, by using the simulator, collisions and singularities can be circumvented.

For the project setup depicted in Figure 8, there is one LBR iiwa robotic arm mounted next to an industrial conveyor belt. As seen, the part travels along the conveyor with previously unknown attributes such as velocity and location. The vision system will detect the object as it moves on the conveyor and calculates grasp positions for robotic retrieval. Once calculated, the robot will take the information and maneuver to pick and place the part in a pre-defined stationary unloading zone. While the RoboDK simulation currently uses a box for a test object, future lab testing will implement objects of various sizes, colors, and shapes.

4. PRELIMINARY RESULTS

Lack of documentation regarding the Roboception cameras made initial setup and implementation of the 3D vision system complicated. Furthermore, despite point cloud renderings and depth images being generated, issues arose in translating 3D visual data into a usable form for KUKA interpretation. Due to lack of compatibility between systems and complexity of implementation, the 3D vision system was abandoned in favor of the promising results found for the 2D vision system.

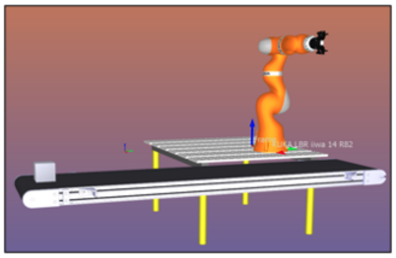
The 2D vision system was evaluated by passing the test object shown in Figure 5 underneath the Pi camera for 60 trials. Successful object bounding boxes and grasp locations were judged by a researcher observing a visual overlay of the camera feed. A successful bounding box was found for 95% of the trials, while an adequate grasp was found for 80% of the trials.

During testing, two variables were identified that had a considerable impact on the accuracy of the system: lighting and object speed. Both were varied during the trials. Additional tests are needed to find the ideal lighting and speed conditions for manufacturing use. Once these are found and implemented, the accuracy of the system is expected to increase.

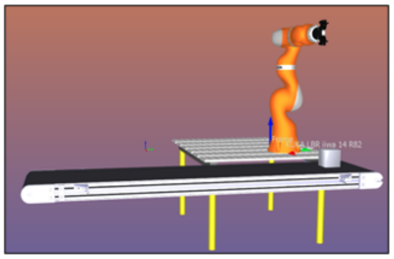
Additionally, it was clear that a camera with a wider field of view would also improve the system performance. While using the Pi camera, objects were within the field of view for less than one second and often had not received an accurate grasp prediction until they were nearly out of view. A wider field of view would allow the vision system to track the object for longer and thus give more accurate predictions.

Some testing was also completed in order to evaluate RoboDK. The connection between RoboDK and the LBR iiwa robot was successfully established by using the RoboDKiiwaInterface [27]. This connection allowed the real robot to follow the simulated robot's positions and commands. In addition, by utilizing the Python programming tool in RoboDK, the robot was programmed to move its TCP to at a certain position by specifying the x, y, and z coordinate and Euler angles. These manual controls emulated the output from the 2D vision system.

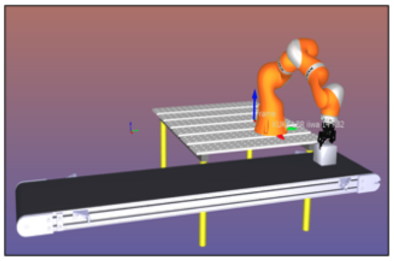
Step 1: Object moves down conveyor.



Step 2: Object reaches robot workspace. Object is detected and grasp positions are calculated.



Step 3: Object is grasped.



Step 4: Object is placed in a pre-defined area.

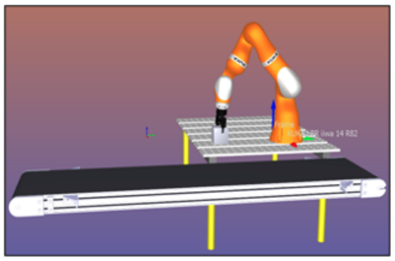


FIGURE 8: ROBODK SIMULATOR

5. DISCUSSION

The promising preliminary tests of the 2D data processing system using a Raspberry Pi show that it has applicable use in industrial applications. Two issues need to be solved before more sophisticated real-world tests can be carried out. First, grasp height needs to be computed. One possible solution to this problem can be modeled after the approach used by [16] where

Mehrez et al. designed a vision system for grasping stationary objects by using a LIDAR sensor to detect the height of each object. We could incorporate this method into our vision pipeline to detect the height of each object at location (gx, gy). Grasp heights can also be found by coupling machine learning models with the 3D data from a point cloud rendering. The determining factor between these two choices will be based on price and desired accuracy. Second, the camera used for object and grasp detection will need to be calibrated in relation to the real workspace. Camera calibration is a widely studied topic and can be accomplished via a variety of techniques [28]. Once these problems have been addressed, the system will be used with KUKA LBR iiwa and KUKA KORE robotic arms to simulate an industrial environment where speed and accuracy tests will be performed.

The communication channel between the vision system and RoboDK still needs to be initiated before testing with a real-world arm can be performed. This can be done by establishing a simple TCP/IP connection between the Raspberry Pi and the host computer with RoboDK installed. Using the RoboDK API, messages can be received and then used to instruct the robot's motion. An overview of the entire system is shown in Figure 9.

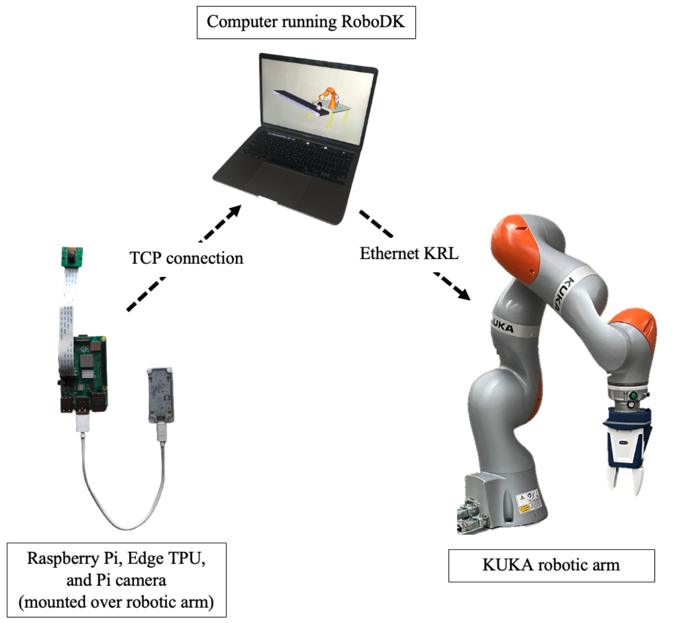


FIGURE 9: OVERVIEW OF CONNECTIONS BETWEEN THE VISION SYSTEM AND THE KUKA ROBOTIC ARM.

6. CONCLUSION

In this work, an automated robotic arm system designed to interact with components in an industrial setting was discussed. Multiple options were explored for the three stages of the framework: (1) visual data capture, (2) data interpretation, and (3) command generation and output to robotic arms. Data capture was investigated in 3D with a Roboception vision system and in 2D with an RGB camera. The examination of this data in 3D used point clouds and in 2D utilized a machine learning model-based algorithm. Finally, commands from these systems can be produced and sent to two different robotic arms using Ethernet KRL. Testing of the systems showed promising initial results for the 2D vision system and RoboDK simulator.

ACKNOWLEDGMENT

The authors would like to thank the University of Iowa Mechanical Engineering Department for their funding and encouragement in the undertaking of this work. This project would not have been possible without them.

REFERENCES

[1] Smys, S., and Ranganathan, G., 2019. "Robot assisted sensing control and manufacture in automobile industry". *Journal of ISMAC,* ***1***(03), pp. 180–187.

[2] Niola, V., Rossi, C., and Savino, S., 2007. "Vision system for industrial robots path planning". *International journal of Mechanics and control,* ***8***(1), pp. 35–45.

[3] Švaco, M., Šekoranja, B., Šuligoj, F., and Jerbić, B., 2014. "Calibration of an industrial robot using a stereo vision system". *Procedia Engineering,* ***69***, pp. 459–463. 24th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2013.

[4] Pan, Z., Polden, J., Larkin, N., Van Duin, S., and Norrish, J., 2012. "Recent progress on programming methods for industrial robots". *Robotics and Computer-Integrated Manufacturing,* ***28***(2), pp. 87–94.

[5] Zhao, Z.-Q., Zheng, P., Xu, S.-T., and Wu, X., 2019. "Object detection with deep learning: A review". *IEEE Transactions on Neural Networks and Learning Systems,* ***30***(11), pp. 3212–3232.

[6] Girshick, R., Donahue, J., Darrell, T., and Malik, J., 2014. "Rich feature hierarchies for accurate object detection and semantic segmentation". In 2014 IEEE Conference on Computer Vision and Pattern Recognition, pp. 580–587.

[7] He, K., Zhang, X., Ren, S., and Sun, J., 2014. "Spatial pyramid pooling in deep convolutional networks for visual recognition". *Lecture Notes in Computer Science*, p. 346–361.

[8] Girshick, R., 2015. Fast r-cnn.

[9] Ren, S., He, K., Girshick, R., and Sun, J., 2017. "Faster r-cnn: Towards real-time object detection with region proposal networks". *IEEE Transactions on Pattern Analysis and Machine Intelligence,* ***39***(6), pp. 1137–1149.

[10] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A.,

2016. You only look once: Unified, real-time object detection.

[11] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C., 2016. “Ssd: Single shot multibox detector”. *Lecture Notes in Computer Science*, p. 21–37.

[12] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C., 2018. “Mobilenetv2: Inverted residuals and linear bottlenecks”. In 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 4510–4520.

[13] Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., and Adam, H., 2019. Searching for mobilenetv3.

[14] Xiong, Y., Liu, H., Gupta, S., Akin, B., Bender, G., Wang, Y., Kindermans, P.-J., Tan, M., Singh, V., and Chen, B., 2021. Mobiledets: Searching for object detection architectures for mobile accelerators.

[15] Morrison, D., Corke, P., and Leitner, J., 2018. Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach.

[16] Mehrez, R., Affes, E., Kadri, I., Bouslimani, Y., Ghribi, M., and Kaddouri, A., 2020. “Location and vision techniques to control a kuka kr6 r900 sixx robot arm”. In 2020 1st International Conference on Communications, Control Systems and Signal Processing (CCSSP), IEEE, pp. 311–316.

[17] Dörfler, K., Rist, F., and Rust, R., 2013. “Interlacing”. In *Rob— Arch 2012*. Springer, pp. 82–91.

[18] Rusu, R., and Cousins, S., 2011. “3d is here: Point cloud library (pcl)”.

[19] Kalal, Z., Mikolajczyk, K., and Matas, J., 2010. “Forward-backward error: Automatic detection of tracking failures”. In 2010 20th International Conference on Pattern Recognition, pp. 2756–2759.

[20] Cass, S., 2019. “Taking ai to the edge: Google’s tpu now comes in a maker-friendly package”. *IEEE Spectrum*, ***56***(5), pp. 16–17.

[21] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D., 2017. Quantization and training of neural networks for efficient integer-arithmetic-only inference.

[22] Lenz, I., Lee, H., and Saxena, A., 2014. Deep learning for detecting robotic grasps.

[23] Depierre, A., Dellandréa, E., and Chen, L., 2018. Jacquard: A large scale dataset for robotic grasp detection.

[24] Hammond, M., 2020. “Robot independent programming system”. PhD thesis, The University of Iowa.

[25] Pieskä, S., Kaarela, J., and Mäkelä, J., 2018. “Simulation and programming experiences of collaborative robots for small-scale manufacturing”. In 2018 2nd International Symposium on Small-scale Intelligent Manufacturing Systems (SIMS), IEEE, pp. 1–4.

[26] Mabrouk, A., Mehrez, R., Bouslimani, Y., Ghribi, M., and Kaddouri, A. “Offline programming with intelligent vision system of kuka robot”.

[27] Safeea, M., 2020. Modi1987/robodkiiwainterface: Drivers for controlling kuka iiwa fron robodk.

[28] Qi, W., Li, F., and Zhenzhong, L., 2010. “Review on camera calibration”. In 2010 Chinese Control and Decision Conference, pp. 3354–3358.