Aerial Detection for Object Placement

Cymberly Tsai

Table of Contents

Abstract

From level ground, human vision is limited to immediate surroundings. For a broader scope of view, humans require a higher vantage point. However, a greater distance sacrifices accuracy. Humans need to efficiently and accurately place objects over large target areas for various tasks such as agricultural mass-seeding, combating wildfires, or marine surface surveillance. To expand human vision for wide-range object placement, the project is a stationary machine designed to autonomously detect a container from an aerial view and complete a simple task: launch a marble into the container. The machine searches for the container with machine learning then estimates the distance from the container and finally calculates the angle to launch the marble. The machine begins centered 25 cm away from the edge of a 1.0 by 0.5m area. The machine detects a 14.2 cm tall container with a 9 cm diameter placed at a random location within the defined area. When the machine locates the container, the machine orientates to launch a 14mm diameter marble into the container. After ten trials, the machine achieved a 90% success rate of landing the marble in the container. Improvements include a more consistent launch mechanism or a more efficient detection model to increase application. Future experimentation includes a larger area or varying targets to further challenge the machine. Ultimately, features of the current machine functioned as intended and could be implemented in more extensive real-world projects.

Introduction

From the ground, human vision is limited to an individual's immediate surroundings. For a greater span of vision, one needs a higher perspective. However, greater distance from a greater height sacrifices the accuracy of human vision. Additionally, human mobility in higher vantage points is limited given the risk of insecure positioning.

A solution is to automate the process, increasing range of vision, increasing accuracy of vision, and reducing risk to humans. Rather than a human going to a high place and looking down, a machine can go to a high place and look thoroughly on the surroundings, and transfer information to humans. With this information from wide-range vision, the machine can go a step further by carrying out a task. Tasks such as mass seeding in agriculture, combating wildfires, or maritime rescue can be done with greater ease and accuracy. The ability to view a large area, to detect objects over a wide plane, is intrinsic to the human ability to expand technological advantages; thus automating the process is the natural course of innovation.

Engineering Goal

Create a machine with machine learning algorithms to autonomously detect a 14.2 cm tall container 9 cm in diameter located within a 1.0 by 0.5 m area and launch a 14 mm diameter marble into the container from a stationary position 0.25 m from the center of an edge of the defined area.

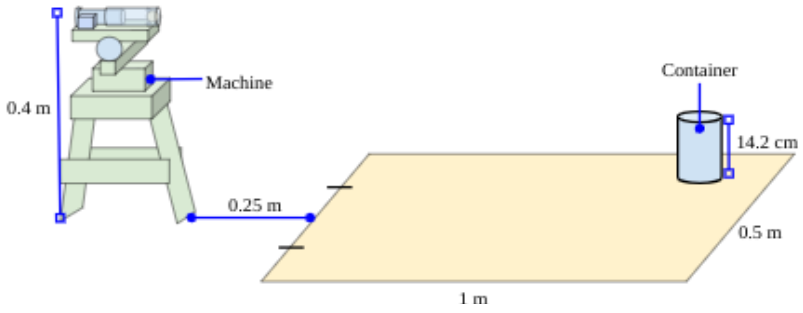


Fig. 1. Diagram of experiment setup

Materials

Hardware

- Raspberry Pi 4
- Servo Motors (SG90, FS90R)
- Stepper Motor (28BYJ-48)
- Breadboard
- Wires/Resistors/AA Batteries
- Webcam (Logitech C310)
- Hollow Cylinder/Elastic

Python Libraries/Packages

- OpenCV
- NumPy
- YOLOv3 - 320

Note: Program written in Python 3.7

System Overview

Overall Approach

The goal of this project can be broken down into three steps: 1) find the container, 2) turn toward the container, 3) launch the marble. Each step corresponds with a machine process. To find the container, the machine needs to detect the container. To turn toward the machine, the machine needs to get the position of the machine. To launch the marble, the machine needs to get the angle for launch. Each step also has mechanical processes associated with each step for the machine to physically carry out.

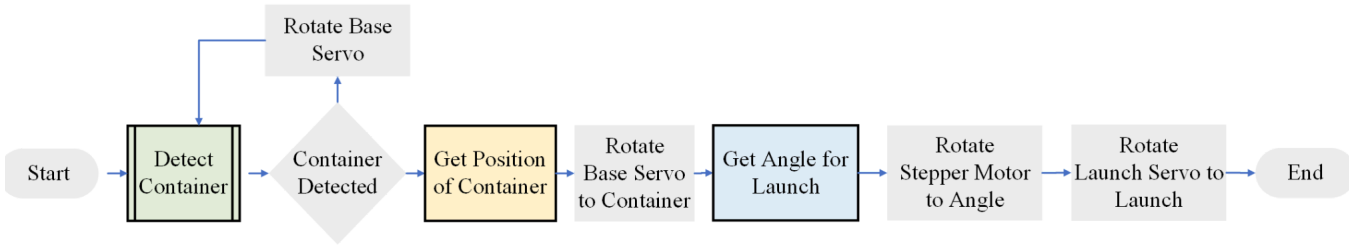


Fig. 2. Flowchart of the machine process

Machine Design

Each computation step the machine performs is ultimately for a mechanical process. For the machine to search for the container and turn toward the container, the machine has a base servo motor for controlled rotation. For the machine to set the angle of the launch mechanism to launch the marble, the machine has a stepper motor. For the machine to launch the marble, the machine has a launch servo motor. And this entire unit is on an elevated platform for an aerial perspective.

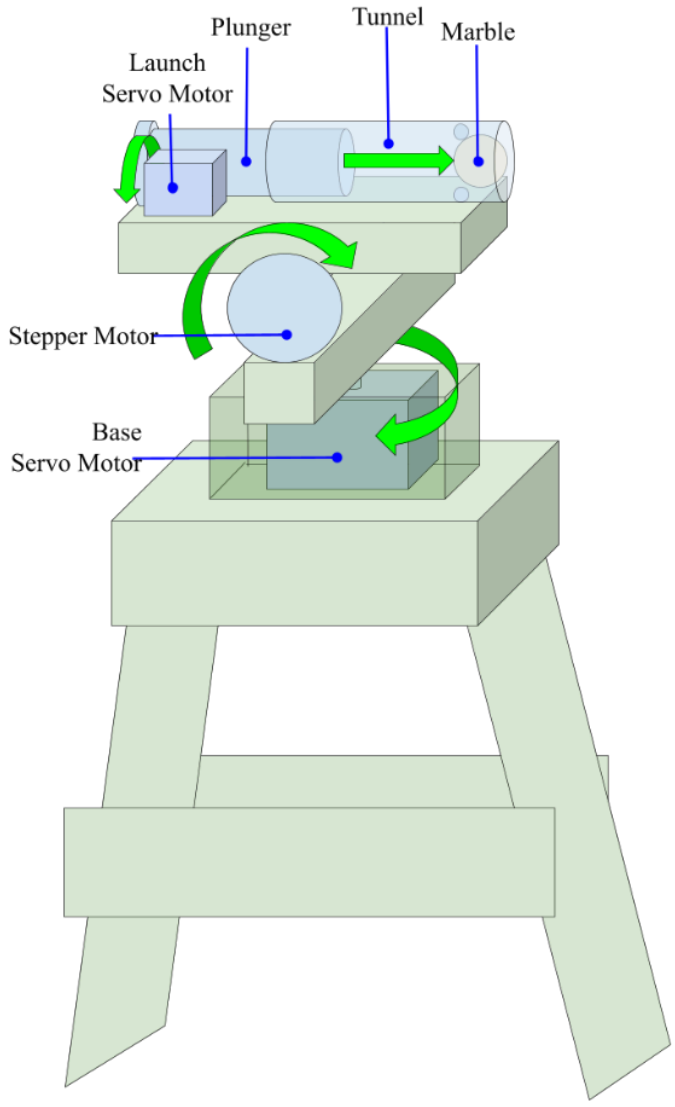


Fig. 3. Diagram of Machine

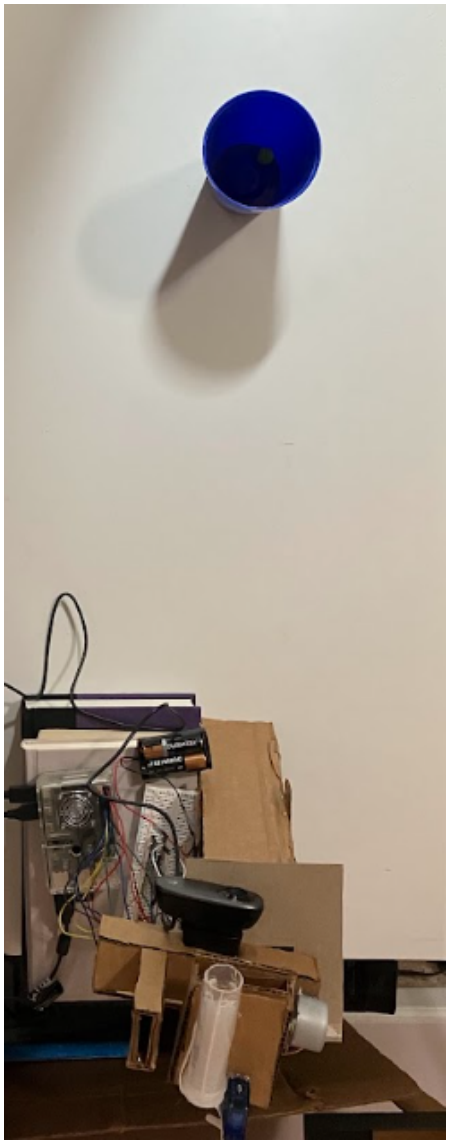


Fig. 4. Image of trial

Detecting the Container

The machine searches for the container by repeatedly rotating the mounted camera on the base script the container is in view. The camera will begin facing the rightmost position and run the detection model. If the container is not in view, the base servo motor will then rotate the base unit, with the mounted camera, approximately 30 degrees, then run the detection script again. The machine will continue to rotate and run detection until either the cup is found or it reaches the final position and stops.

The Detection Script

The primary purpose of the detection script is to determine if the container is in view, if the container is in view. The detection script implements the You Only Look Once (YOLO) model, an open source, pre-trained machine learning model for object detection, classification, and localization. This project used YOLO version 3 accepting images 320 x 320 px (YOLOv3-320), compatible with RaspberryPi 4. The name, “You Only Look Once”, refers to how the model processes the entire image at once. The model applies a layer that divides the image into cells (Fig. 6) for the program to process each cell and determine consistencies of an object. The model then identifies objects, along with classification and bounding boxes, from various processing layers (Fig. 7). Finally, the model suppresses repeated references to the same object (Fig. 8). Given the results from the YOLO model, if the container is in the given image, the program returns the width and center coordinate of the container in the image.

Fig. 5. Original Image

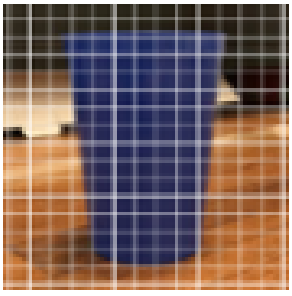


Fig. 6. Divided into cells

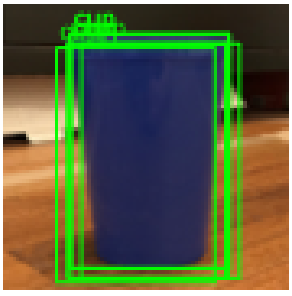


Fig. 7. Object detected

Fig. 8. Suppressed redundancies

Getting the Position of the Container

Location of Container Relative to Machine

This process entails the container's position on the horizontal axis relative to the machine. Given the center coordinate of the container detected from the detection portion, the program compares the center x-coordinate value to a center range of 50px in the center of the image. If the center x-coordinate falls outside the 135px and 185px range of the image, the program will determine if the base should be rotated clockwise or counter-clockwise to face the container. Once the container is considered centered, the machine can proceed to determine the distance of the container from the machine.

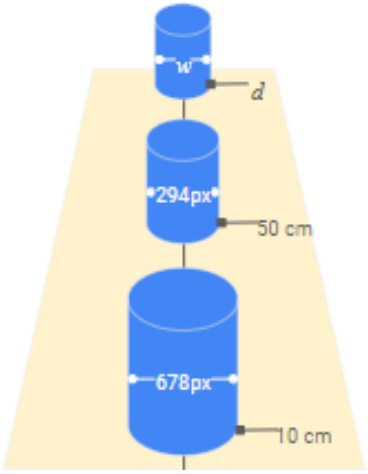


Fig. 10. Model of width-distance relationship

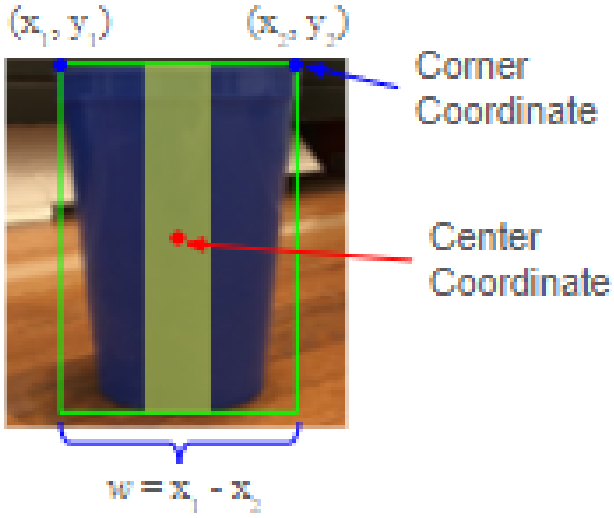


Fig. 9. Annotated image of reference points

Distance of Container Relative to Machine

The machine needs the distance of the cup from the machine to determine the angle of launch. The methodology for determining the distance is taking advantage of the relationship between perceived width of the container in pixels in the image and the distance of the container from the machine in centimeters (later converted to meters for calculations). Data was collected for the

relationship to generate an equation. The container was placed at distances 25-125 cm, at every ten centimeters. At each position the width of the container, as detected by the machine, is measured 100 times. The average width at each position is then taken to form a dataset of widths(px) vs distances(cm), which is then used to generate an equation using the Python NumPy package in form of $d = Aw^2 + Bw + w$ where d = distance and w = width. Once the container is approximately centered in the camera view, the program passes the width from the detection script to the equation and saves the calculated distance as a variable for later use.

Getting the Angle for Launch

Initial Velocity (v_o) from Launcher

To determine the angle of launch, the initial velocity of the launcher must be known. The initial velocity was calculated from 11 trials of launching the marble at a fixed angle of 35° angle. For each launch, the distance was measured. Given distance (Δx) and angle (θ), an equation was derived for initial velocity (v_o). The average velocity was calculated to be 5.783 N.

Launch #	Distance (m)	V_o (N)
1	0.91	5.850
2	0.88	6.127
3	0.90	6.235
4	0.93	5.976
5	0.90	6.102
6	0.87	4.965
7	0.85	6.071
8	0.84	6.151
9	0.89	5.515
10	0.89	5.596
11	0.87	5.030

Table 1. Launch trials for distance and velocity

$$\Delta x = v_x t + at^2$$
$$v_x = v_o cos\theta$$
$$\Delta x = v_o cos\theta * t$$
$$v_o = \frac{\Delta x}{t cos\theta}$$

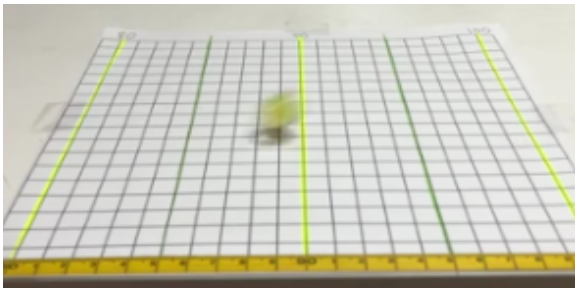


Fig. 11. Measuring distance of launch

Calculate Angle for Launch

Given distance (Δx) and initial velocity (v_o), solve for launch angle(θ).

$$v_{max\,y} = v_{initial\,y} + gt_{max}$$
$$v_y = v_o sin\theta$$
$$0 = v_o sin\theta + gt_{max}$$
$$t_{max} = \frac{-v_o sin\theta}{g}$$
$$t_{total} = \frac{-2v_o sin\theta}{g}$$
$$\Delta x = v_o cos\theta * t$$
$$\Delta x = \frac{-2v_o^2 sin\theta cos\theta}{g}$$
$$-\frac{g\Delta x}{v_o^2} = 2sin\theta cos\theta$$
$$-\frac{g\Delta x}{v_o^2} = sin2\theta$$
$$\theta = \frac{sin^{-}\left(-\frac{g\Delta x}{v_o^2}\right)}{2}$$

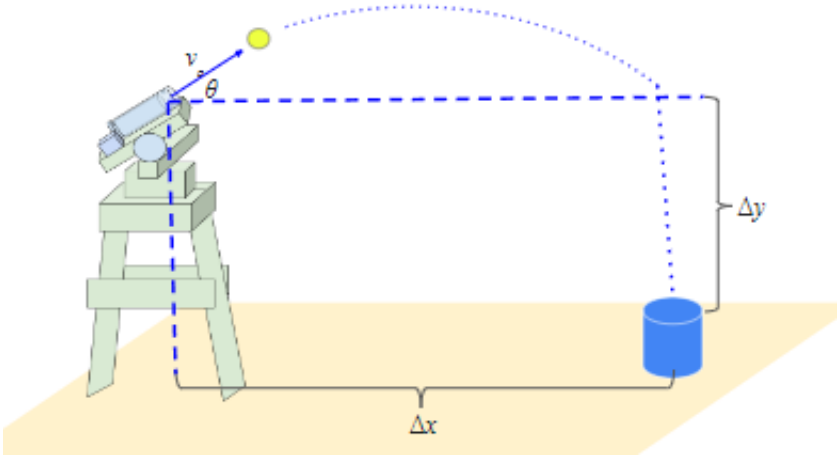
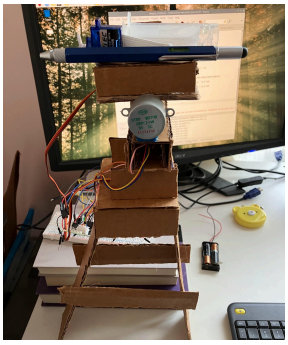


Fig. 12. Diagram of projectile

Angle calculated and saved as a variable. Program then rotates the stepper motor toward angle.



F.g. 13. Side view of machine

Launch Mechanism

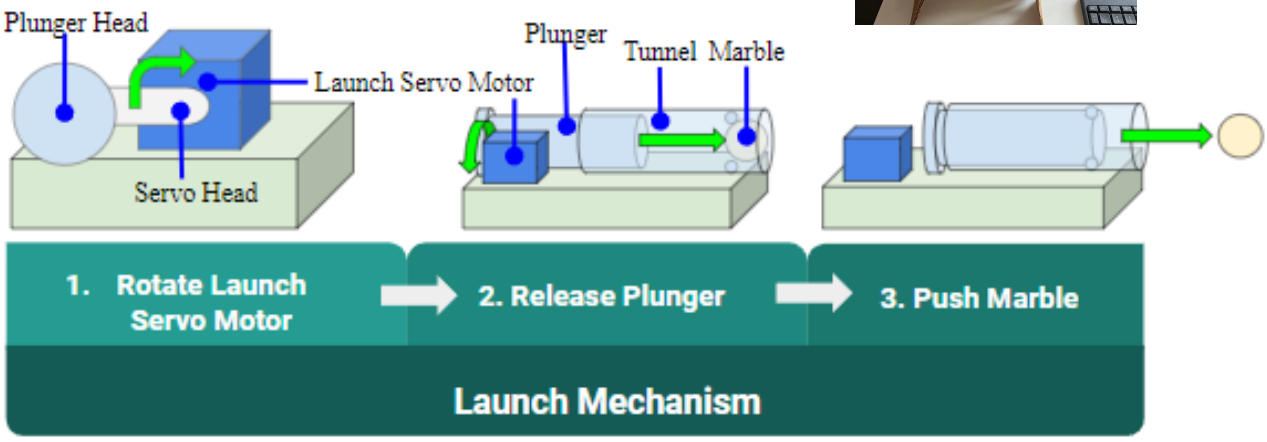


Fig. 14. Diagram of launch mechanism

Procedure

1. Place container at random location within 1.0 by 0.5 m. area
2. Place machine 0.25m away from center edge of area
3. Turn on machine, run program, and allow machine to move
4. When marble is launched, record if marble is in container (true/false)
5. Repeat steps 1 through 4 ten times

Results

The machine achieved a 90% success rate, failing 1 out of 10 trials.

Trial Number	1	2	3	4	5	6	7	8	9	10
Contact with Target	True	True	True	True	False	True	True	True	True	True

Table 2. Boolean result of each trial.

Analysis and Conclusion

The machine was a success with a passing success rate. The failure was caused by an inconsistent launching mechanism where the marble fell out of the launch tunnel prior to launch. Regarding the launch mechanism, the machine would also benefit from a method of launch with a standard initial velocity. During the testing for the initial velocity of the launch mechanism, the magnitude of velocity varied from the mean with a standard deviation of 0.450 N, which could be reduced by more consistent launch mechanisms such as an alternative elastic or alternative mechanism for applying force to the marble. As the target had a relatively large diameter, the experiment was fairly forgiving to the machine, so slight variations in initial velocities between trials would not have much of an impact on the results.

Another limitation of the machine was the time consumption, as each trial took 2 to 5 minutes to complete. A bulk of the time was used for running the detection model, as the machine stopped to run the detection model for roughly 5 seconds at each position. The machine's process could have been expedited with continuous, real-time object detection so the machine could continuously move without stopping. Another aspect of the detection model is that the YOLOv3 model was specifically chosen to run on the RaspberryPi, which has a limited capacity for processing, given the focus on mobility over processing power. An alternative computer processing unit with greater capability for processing could handle a faster machine learning model for continuous, real-time object detection.

An additional potential improvement is for the turning mechanisms, for both the angle determining motor and the base rotational motor. Should the machine need to have a more specific target with a smaller area, the turning mechanisms would need greater precision for an accurate alignment of the machine to the container and setting the angle for launch. For example, a stepper motor could have been used in place of the servo motor as the base rotational mechanism, allowing for greater precision for rotation toward the container.

Further general advancements could be made to the construction of the machine. As the machine was largely structured with cardboard, the machine would not last long in real-world application. A more durable material specifically cut for a design similar to the machine of the project would have greater success in modern application.

With these improvements in mind, further investigation could be run in a larger area to test a greater range of object detection and placement. Other experimentation could also include varying targets, such as multiple or moving targets. The machine could also be redesigned to incorporate alternative inputs for search such as infrared or sonar as alternative methods of object detection. Future

experiments should also include an increased number of trials, rather than the simple ten this project provided.

The project was an overall success, meeting the set engineering goal with a passing success rate. Though the machine was designed for a very simple task in a controlled environment, elements of the machine could be further implemented in larger, more complex projects for more meaningful tasks. Real-world applications of wide-range object detection and placement already exist, such as mass seeding or marine surveillance, however these options could be improved upon with the approach described in this project. Looking to the future, additional applications for an aerial approach to object placement could be developed with the progression of technology to expand the capabilities of technology in our world.

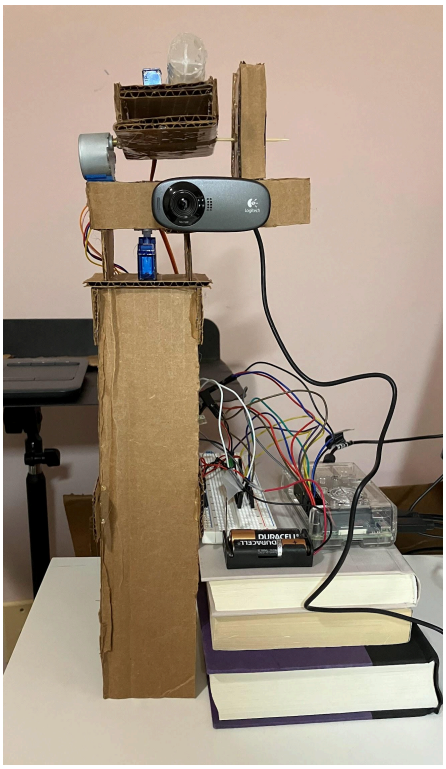


Fig. 15. Front view of machine

Fig. 16. Side view of launch mechanism

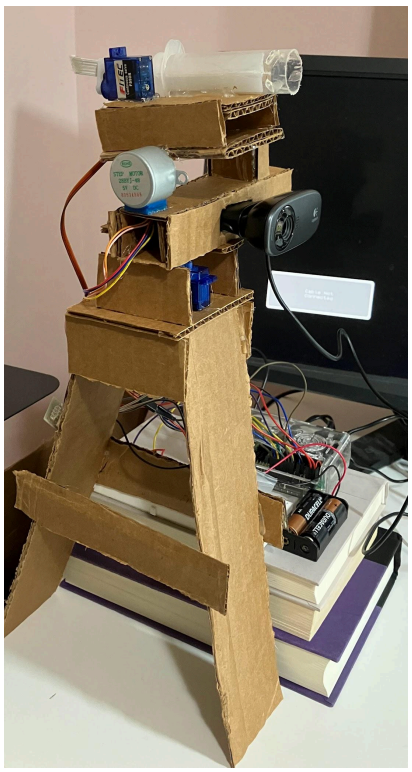


Fig. 17. Slant view of machine

References

Brownlee, Jason. "4 Distance Measures for Machine Learning." Machine Learning Mastery, Mar. 2020, machinelearningmastery.com/distance-measures-for-machine-learning/. Accessed 5 Nov. 2023.

“Servo Motor SG-90.” Components 101, 18 Sept. 2017, https://components101.com/motors/servo-motor-basics-pinout-datasheet. Accessed 8 Nov. 2023.

"Raspberry Pi Documentation." Raspberry Pi, www.raspberrypi.com/documentation/computers/os.html. Accessed 1 Nov. 2023.

Raspberry Pi Pin Layout. Raspberry Pi, www.raspberrypi.com/documentation/computers/os.html. Accessed 1 Nov. 2023.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 779–788). https://doi.org/10.1109/CVPR.2016.91

Saha, Sumit. "A Comprehensive Guide to Convolutional Neural Networks - the ELI5 Way." Towards Data Science, 15 Dec. 2018, towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. Accessed 5 Nov. 2023.

Appendix A

Program Pseudocode

```
Import files and packages
Initialize hardware
While True:
   While detector returns False
      If servo not at final position then
         Rotate base servo
      Else return “no container found” and break
   While detector returns center coord outside center range
      Rotate base servo
   Calculate distance from container
   Calculate angle needed for launch
   Rotate stepper to angle
   Rotate launch servo to launch
Clean hardware
```

Appendix B

Partial Program for Detection Script

```
# Detection Script
import cv2
import numpy as np

classes = []
with open('coco.names', 'rt') as f:
    classes = f.read().rstrip('\n').split('\n')
net = cv2.dnn.readNetFromDarknet('yolov3-320.cfg',
'yolov3-320.weights')
net.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)
net.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU)

def getImage():
    cam = cv2.VideoCapture("/dev/video0")
    cam.set(cv2.CAP_PROP_FOURCC,
cv2.VideoWriter_fourcc('M','J','P','G'))
    success, img = cam.read()
    newImg = cv2.dnn.blobFromImage(img, 1 / 255, (320, 320), [0, 0,
0], 1, crop=False)
    net.setInput(newImg)
    outLayer = []
    for n in net.getUnconnectedOutLayers():
        outLayer.append(net.getLayerNames()[n - 1])
    outputs = net.forward(outLayer)
    cv2.waitKey(0)
    return [outputs, newImg]

def runDetection(outs, img):
    box = []
    ide = []
    conf = []
    mid = []
    for a in outs:
        for b in a:
            scores = b[5:]
            classId = np.argmax(scores)
```

```
            con = scores[classId]
            if con > 0.4:
                w = int(b[2] * 640)
                h = int(b[3] * 640)
                x = int(b[0] * 640 - w / 2)
                y = int(b[1] * 480 - h / 2)
                box.append([x, y, w, h])
                ide.append(classId)
                conf.append(float(con))
                mid.append([int(b[0]*640), int(b[1]*640)])
    keep = cv2.dnn.NMSBoxes(box, conf, 0.4, 0.3)
    results = []
    for k in keep:
        results.append([classes[ide[k]], box[k][2], mid[k]])
    return results

def returnResults():
    image = getImage()
    results = runDetection(image[0], image[1])
    if len(results) > 0:
        for t in results:
            if t[0] == "cup":
                return results[results.index(t)]
    return False
```

Appendix C

Program Partial Code for Position Calculations

```
# Data
w = [...]
d = [...]
eq = numpy.polyfit(w, d, 2)

# Conversion for Distance
distance = eq[0]*width**2 + eq[1]*width + eq[2]

# Calculation for Angle
angle = math.asin(9.8*distance/math.pow(5.781,2))/2
```