Computer Vision Catch Prediction Robot

Ray Hu*
CSE 598: Advances in Robot Learning
Arizona State University
Tempe, USA
rhhu1@asu.edu

Jacob Lusenhops*
CSE 598: Advances in Robot Learning
Arizona State University
Tempe, USA
jlusenho@asu.edu

I. Introduction

Computer vision is a staple component within robotics. With the advent of more sophisticated learning techniques and improved hardware, cameras are one of many ways that allow robots to see and interact with their environment. Our project is a simple and clear example of this, combing both software and hardware tasks through the medium of a camera.

A. Problem

Our task to solve is in the realm of kinematics. We aim to predict the landing zone of a projectile in motion on a single axis with a camera, and with this prediction, we hope our robot would be able to move accordingly to catch it. On paper, this seems easy enough with the established rules of kinematic motion, but there is a twist to our setup environment. Instead of facing the camera perpendicular to the axis of motion, we are facing the camera towards, which introduces more challenges to overcome.

B. Challenges to Solve

There are two types of challenges that must be addressed in this task: **software** and **hardware**.

1) Software: As aforementioned, most examples of kinematic motion in basic physics are perpendicular facing to the frame of reference. When facing parallel to an object in kinematic motion, there are no easy rules or formulas to figuring out where it is going to land. In such complex situations, we turn to statistical methods, specifically machine learning. By capturing the object's features in camera space, we can train a model to predict where it will land. This introduces our first challenge. A model's performance is only as good as the data it is given. Careful data collection must be established to have the best outcomes during training. Moreover, while solving this may accomplish our goal as just a computer vision task, robotics and its connection to hardware introduces more challenges. Within this realm, time constraints are an important requirement to address. Data recording and model inference must be completed within a certain time to allow the robot to make adjustments before the object hits the ground. Therefore, a careful balance of performance and computational efficiency must be established.

[0]* These authors had equal contribution

2) Hardware: Within robotics, hardware represents the ins and outs of information. For this task specifically, we require our robot to receive the best possible data from our cameras while also requiring our robot to be fast in its movements. This introduces the two challenges of speed and consistency. The camera must be fast enough to capture all data points to be sent to the model while the robot must be fast enough to act upon the predictions given by the model. Essentially, we are racing the time it takes for an object to fall. Furthermore, this all must be done consistently with the same accuracies every time. If there is always some degree of error on the movements of the robot or the camera capture speed, then results will guaranteed to always be different.

C. Equipment

The hardware equipment used for this project include

- Belt driven actuator
- Arduiono
- Webcam
- Ping pong ball
- Cup

II. Methodology

A. Hardware Setup

We first introduce the hardware setup for our project. This includes listed items from the equipment subsection and the interactions between them, as shown in Figure 1 and 2.

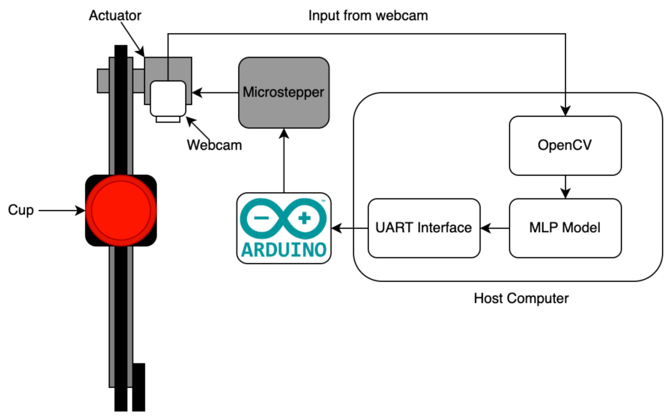

Fig. 1. Diagram of Hardware components and their interactions

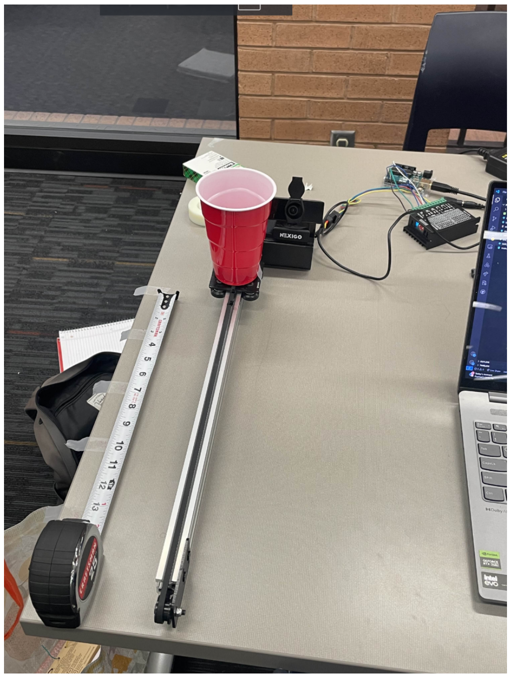
Fig. 2. Hardware components in actual experiment

Video is gathered using the webcam and sent to the host's computer. From there, various software programs are used to extract features of the ball, create a prediction using the machine learning model, and communicate the prediction to the robot. Within the robot, this prediction is passed through an Arduino, a micro-stepper, and finally to the actuator.

B. Software Setup

On the software side, multiple scripts and libraries were used to help accomplish our goals. The first portion includes OpenCV. We first had a color range detector script from [1] to accurate get the HSV color range of the object we wish to track. Using this color range and the baseline ball tracking script from [1], we developed a script to track our object in motion with our webcam. The next step is the machine learning model portion with PyTorch. In this part, we developed various model architectures to train and test. After evaluating them, we choose the best ones to test with our hardware components. To connect our model with these components, we have the UART interface with the serial package. This allows us to send our model prediction to our robot via a USB cable.

C. Model Architectures

We developed a multitude of architectures. Listed here are the most significant ones that was tested with our hardware.

1) Multilayer Perceptron (MLP): This is a basic four layer feed forward neural network with an input layer, two hidden layers, and an output layer. Figure 3 shows our model used.

2) Recurrent Neural Network (RNN): Since our data falls in the space of time-series analysis, we tested a RNN structure to better handle this kind of data.

3) Convolutional Neural Network (CNN): This was a non-traditional technique where we embedded our input data into an input shape of 224 x 224 x 3 and ran it through a pre-trained ResNet 18 model.

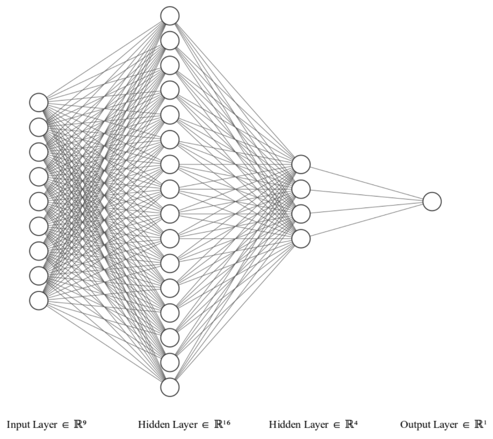
Input Layer $\in \mathbb{R}^9$
Hidden Layer $\in \mathbb{R}^{16}$
Hidden Layer $\in \mathbb{R}^4$
Output Layer $\in \mathbb{R}^1$

Fig. 3. Our MLP model

D. Data Collection

We collected our data using a two webcam setup. Our primary webcam is faced on the axis of motion, and its purpose is to extract the position and radius of the ball in camera space for a set number of time steps. This is what we will use for data in our samples. Next, a secondary camera is faced perpendicular to the axis of motion, and its purpose is to find where the ball will land for our sample's target values.

E. Model Training

To build and train these models, we used PyTorch as it allowed us to easily integrate our models in our python environment. During this process, we use the data we have collected and evaluate the training and validation loss.

III. Results

While there is room for improvement, our experiments showed promising results. The final trained model achieved a mean squared error (MSE) of 0.0398 on an output scaled from 0 to 1. This loss is not ideal, but it does still show a strong correlation and that the model was able to pick up on differences between throws. Figure 4 shows this loss over epochs.

In terms of efficiency, the average time it took to collect the number of frames needed for inference was 0.1 seconds, while the time it took to perform inference was 0.004 seconds. The combined data collection and inference times were very close to the maximum allowable time to make a guess and still have time to move the cup. The inference makes up a small percentage of the total time to perform a guess, so if a larger model was used to achieve a better loss, time would most likely need to be made up in other parts of the system.

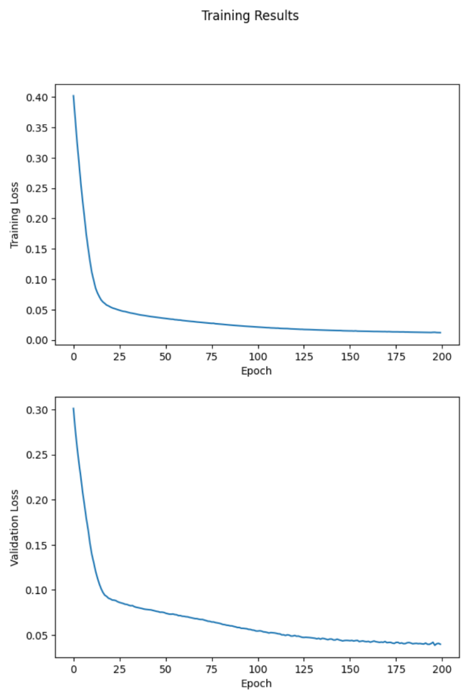

Fig. 4. Training and validation loss over epochs

IV. Limitations and Future Work

Several factors constrained the overall performance. Due to the strict latency requirements, the model was kept small, limiting the maximum performance of the model. Furthermore, the camera used was only 30 Hz which limited the number of frames that could be collected. With a faster camera, the resolution could be increased allowing the model to detect more subtle differences in throws. Similarly, improving the actuation time could allot more time to inference and improve performance.

This setup was limited to a single dimension, but in the future, a second actuator could be added perpendicular to the current one to make the setup two dimensional.

V. Conclusion

This project showed that it is feasible to predict the landing position of a ball in real time along a single axis. By combining computer vision with a small neural network, we were able to achieve low-latency predictions with reasonable accuracy. Despite the many constraints, the system still was able to function effectively. For future work, improving the camera, actuators, and model could further improve accuracy.

Here is a link to our result video https://youtu.be/jMWDZReYIyc

References

[1] A. Rosebrock, "Ball tracking with opencv," PyImageSearch, https://pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/ (accessed May 9, 2025).