Robotic Chess Player Arm and Computer Vision Oriented Object Detection with Raspberry Pi

CSE237A Final Project Report

Jiawen Xu, Jamshed Ashurov

April 12, 2023

1 Abstract

In this project, we design and implement an intellectual robotic arm that can play chess with humans. The robotic arm is controlled by Raspberry Pi. We build the network communication between Raspberry Pi and our laptop through socket programming in Python. The laptop receives signal from camera and performs computer vision technique to provide control and prediction to the robotic arm. Once the Raspberry Pi receives the signal, it moves accordingly by setting duty cycle in PWM. In our Project, we successfully implements the application of embedded system, machine learning, and network communication.

2 Team members

Jiawen Xu

- Email: jix034@ucsd.edu
- PID: A16529903

Jamshed Ashurov

- Email: jashurov@ucsd.edu
- PID: A15475198

3 Motivation

Developing an intellectual robotic arm requires a multidisciplinary approach including electrical engineering, computer science, and artificial intelligence. Robotic arms increases productivity which can perform tasks much faster and more efficiently than human being if it is well designed. For dangerous environment, usage of robotic arms can reduces of the risk of human workers getting in dangers.

We chose to design a robotic chess player arm for our projects as it combines knowledge from embedded systems on both software and hardware sides as well as artificial intelligence or reinforcement learning. This project will provide us a good chance to build the bridge of interaction between software and hardware.

4 Related work

A similar project was done in *Autodesk Insturctable* where a robotic arm for playing chess was designed.[6] The project posted was a graduation project by a student at Kuwait University-College of Engineering & Petroleum. The robotic arm presented used reed switches to sense the placement of the chess and used conditional software logic to calculate the next movement after the signal is transmitted to the micro-controller.

Ashish Kumar Gupta et al. presents salient object detection (SOD) with computer vision techniques [2]. In their work, they discussed the conventional and deep learning ways to implement computer vision techniques.

Guptas's work shows a possible solution to improve the object detection of the existed robotic arm project, which uses reed sensors.

Instead of using reed switches, in our project, we will use camera and computer vision techniques to provide more accurate sensing of the positions of the chess pieces. Computer vision algorithms can detect the pieces' color and shape, which can be used to determine their positions on the board. It eliminates the usage of reed sensors, which higher the accuracy at detection and efficiency.

The chessboard detection part of this project is one of a kind project, that integrates multiple computer vision projects that are available right now. The framework of our code is based on the paper and code made by David Mallasen[5] which varies CNN models to train and detect the chess board and the chess pieces. What we added to the baseline code is the integration of the chess engine called Stockfish that was largely inspired by

the project made by Anton Elmiger[1]. Moreover, our project consists of network communication between the camera, board detector, and the robotic arm.

5 Component List

5.1 Hardware

1. Adeept 4 Axis Robotic Arm Kit * 1
2. Camera*1
3. Laptop*1
4. CanaKit Raspberry Pi 4 Extreme Kit - 128GB Edition (1GB RAM) *1
5. Arm HAT*1
6. SG90 Servo*3
7. AD002 Servo*1
8. Screws, nuts, jumper cables
9. Chess board and chess

5.2 Software

1. Raspberry Pi OS (image: 2022-09-06-raspios-buster-armhf-lite.img.xz)
2. Apple macOS
3. Programming Language: Python
4. Python Library: OpenCV, RPi.GPIO, Adafruit_PCA9685, time, socket, threading
5. Runtimes and Models: ONNX, tensorflow, keras, NASNetMobile
6. Stockfish-chess AI engine

6 Hardware and Software Integration

6.1 Robotic arm

6.1.1 Hardware

Figure 1 and Figure 2 are the robotic arm we are using for this project. The robotic arm is based on Rapsberry Pi based. The Raspberry Pi is connected to the PCB (ARM HAT) provided with the robotic arm, which transmit the PWM signals through the GPIO pins to the motor (servos) of the robotic arm.There are four rotational motors in the robotic arm that control the motion.

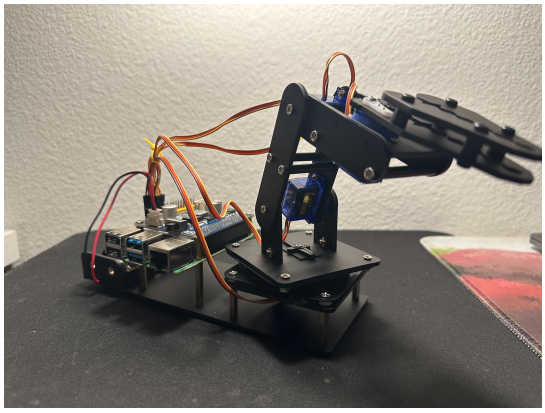

Figure 1: Image of robotic arm

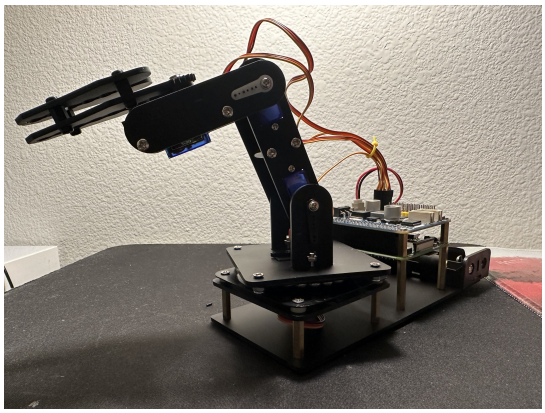

Figure 2: Image of robotic arm

Figure 3 presents the ARM HAT of from the robotic kits, and Figure 4 presents the schematic of the board. In our project, we will only use the Servo control of the board, which maps to the GPIO pins of Rapsberry Pi.

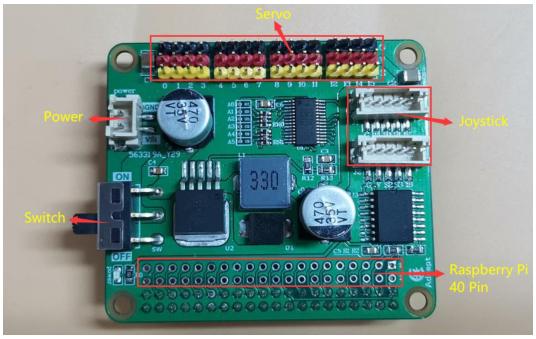

Figure 3: Picture of ARM HAT

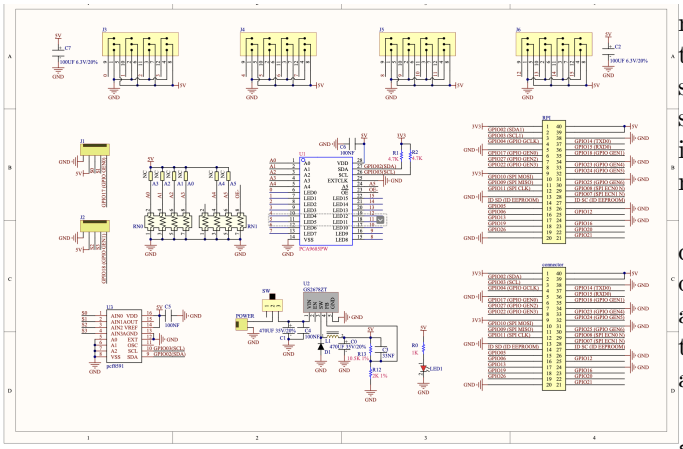

Figure 4: Schematic of ARM HAT

Our robotic arm consists of 4 servos which in charge of the movement of the robotic arms. The servos is labelled 1, 2, 3, 4 in our implementation, and it is programmed by setting the duty cycle and frequency of PWM. Figure 5 presents a picture of a single servo, and figure 6 presents how 4 servos are intergrated into the robotic arm. We will program in software to control the servo for the movement.

Figure 5: Picture of Servos

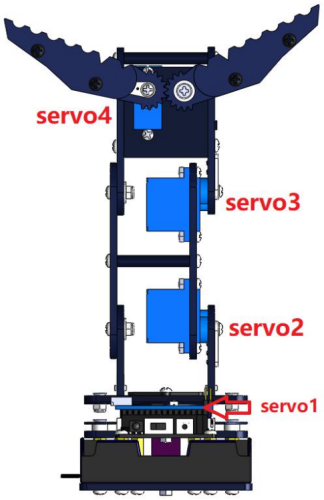

Figure 6: Picture of Servos in robotic arms

6.1.2 Software

The library we use for robotic arm is Adafruit_PCA9685 and RPI.GPIO. Adafruit_PCA9685 is provided by the manufacturer, which behaves as an interface to control the movement of the servos. We control the servos by sending signal through Raspberry Pi using PWM. We set the PWM signal to be 50Hz. The PWM module is imported from Adafruit_PCA9685 library used to communicate with PCA9685.

After initialize the frequency, we control the duty cycle of the PWM for the motion of the robotic arm. The duty cycle varies from 100 to 560 which corresponds to a rotation range from 0 to 180 degree. More specifically, the relation between duty ratio and movement angle is angle $=$ (duty ratio $-$ 100)/2.55.

To control all the servos, we use the function `set_all_pwm(0, duty_ratio)`. To control a specific servo, we use the function `set_pwm(id, 0, duty_ratio)`

The robotic arm receives signal from the laptop through socket programming in python. Whenever the robotic arm receives the instruction from server, the robotic arm will start moving based on the instruction sent from the laptop. The laptop will send a 2-dimensional coordinate, and the robotic arm will query the hard-coded mapping table and retrieve the movement parameters. Then it will keep moving until it reaches the expected position.

6.2 Chess Board Detection

Trial 1: Magnetic chess pieces detection using reed switches

Initially, for this project our idea was to have reed switches for each of the 64 board cells and detect whether a piece of placed or taken. We implemented this first by drilling the holes inside the center of the cells to place the reed switches as shown in figure 7. Afterwards, we connected an LED to each switch which in turn would signal if a chess piece was place on top of it by detecting the magnetic wave of the piece. When we tested the switches by moving the pieces onto and off the board, we noticed a lot of false positive and negatives as shown in figures 8 and 9.

Figure 7: Picture of the reed switch setup

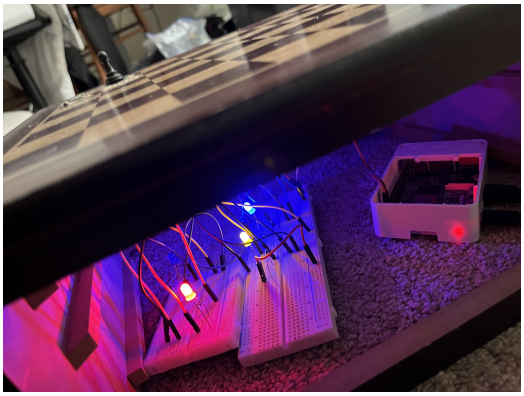

Figure 8: Picture of false positives

Figure 9: Picture of false positives

Trial 2: Chessboard detection a camera with computer vision on RPI4B

Next, we wanted to try to use a camera to detect the chessboard and the chess pieces. It is currently impossible to use the standard version of tensorflow engines on RPIs. thus we used the lightweight version of tensorflow that was specifically designed for RPIs and ARM architecture from this repository[3].

For the baseline design of the chess board recognition, we used the paper and code from Maciej A. Czyzewski[4]. We changed the bits and pieces of the code accordingly to make it work on the RPi version of tensorflow. We were able to successfully detect the chess board corners as shown in figure 10 and 11. However, when we tried adding the training code for chess piece recognition, the raspberry pi crashed(figure 12) which we assume was caused by the intensity of the workload.

Figure 10: Picture of the chees board

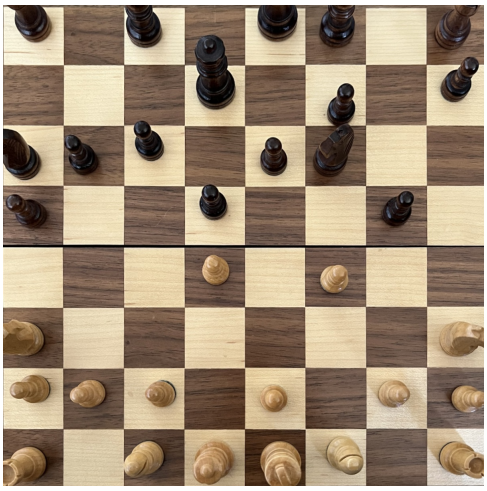

Figure 11: Picture of the detected board corners

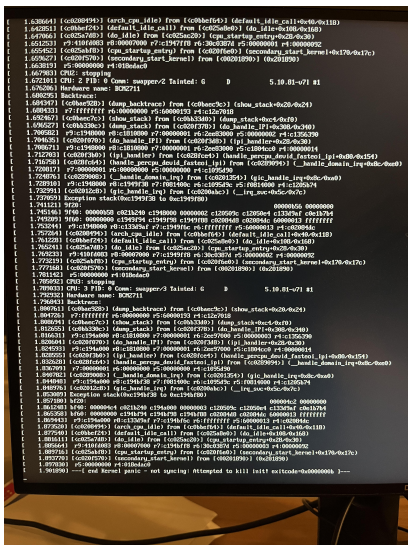

Figure 12: Picture of the raspberry Pi crash log

Final Version: Chessboard detection using RPi for taking the image and MacBook Pro 2016 for processing the image In our final design we established a wireless communication between the RPI and the Macbook to transfer the real time snapshots of the chessboard for processing. For the baseline design of the

chess board recognition, we used the paper by David Mallasen[5]. We also integrated the chess AI engine called Stockfish that would validate the moves and give us the best move to make by analyzing the board. We largely used the project by Anton Elmiger[1] to achive our goal. Lastly, we used Python sockets API to communicate with the devices. The following link contains setup and the demonstration of the devices and communication.

6.2.1 Hardware from all trials

- Magnetic chess board
- 20 Reed Sensors
- Breadboard, LEDs, resistors
- Drill
- Picamera OmniVersion OV5647
- Raspberry Pi 3B
- Camera Stand

6.2.2 Software from all trials

- Python
- Tensorflow, Keras
- ONNX runtime
- CNN Processing models:
  - AlexNet
  - DenseNet
  - MobileNet
  - NASNetMobile
  - SqueezeNet
  - Xception
- Python socket API
- Stockfish AI chess engine

6.3 Challenges

The biggest challenge we encounter in this project is the network hardware constraint that we met.

Regarding the hardware constraint, for robotic arm, we could not make finer movement because of the structure of the robotic arm. When multiple chess is placed together, the other chess will be pushed away from the movement of grabbing the targeted chess.

Also, when running the AI/Computer Vision program, some hardware failure occurs after getting the result. In that case, we have to run the machine learning code on the personal laptop, where a 3-way communication for data transmission is needed.

Network configuration appears challenging due to the instability of the public school network. Luckily, during the demonstration, we are able to access a stable local area network, so we manage to build the communication between the two different modules through network.

7 Experimental Section

7.1 Experimental Setup

For our experiment, we tested the robotic arm and object detection separately. The metrics we were using are response delays and target accuracy. Our latency is defined as the time frame from the point where robotic arm starts moving to the point where it stops moving. For robotic arm, we ran 4 workload with different position. The workloads were grabbing and putting the chess from one position and another position. The expected latency is calculated based on relative distance of starting and ending positions. The accuracy rate will depends on how far the actual destination is from the center of the expected destination. The further it is, the less accurate it is. Some approximation has been made while evaluating the accuracy of the placement.

For the chess board detection, we mainly tested which model has the most accuracy and least latency on both Tensorflow and ONNX runtimes. In this section we will show the top 4 best models that have good accuracy and small latency. The list of all the processing models are in section 6.2.2.

7.2 Robotic Arm

workload	Start	End	Distance
1	D3	E3	1
2	C3	F3	4
3	C2	F2	4
4	B2	G2	6

Table 1: Configuration of Robotic Arm Experiment

Workload #	Measured Accuracy	Measured Latency	Expected Accuracy	Expected Latency
1	80%	6 s	70%	6 s
2	60%	6.5 s	50%	8 s
3	60%	8 s	50%	8 s
4	50%	10 s	50%	12 s

Table 2: Results of Robotic Arm Experiment

From the data presented, we notice that the robotic arm isn't quite accurate on moving chess. We think that is because of the hardware constraint as we discussed before. To better solve this problem, a better designed and more expensive robotic arm might need to be purchased. However, the robotic arm we developed is still considered a functional robotic arm for moving chess around.

7.3 Object Detection

workload	CNN Processing Models
1	Xception
2	NasNETMobile
3	MobileNet
4	DenseNet

Table 3: Configuration of Object Detection Experiment

7.3.1 Tensorflow

Workload #	Measured Accuracy	Measured Latency
1 Xception	92%	14.50s
2 MobileNet	91%	6.84s
3 SqueezeNet	86%	5.77s
4 NASNetMobile	97%	9.06s

Table 4: Results of TensorFlow

7.3.2 ONNX

Workload #	Measured Accuracy	Measured Latency
1 Xception	95%	6.9s
2 MobileNet	91%	1.87s
3 SqueezeNet	89%	1.31s
4 NASNetMobile	95%	2.58s

Table 5: Results of ONNX

As you can see having ONNX as the runtime is much faster and most of the time more accurate than using tensorflow and keras. In our project, since accuracy is more important we used NASNetMobile because it was generally more accurate than MobileNet but also very fast.

8 Conclusion

In the project, we successfully program the movement of the robotic arms and object detection using computer vision. As well, we built the network communication between Raspberry Pi and laptop. Based on our experimental result, we can see that using computer vision for object detection has higher accuracy comparing to the reed switches, and the application of robotic arms reflects its feasibility in reality.

This project consists of embedded hardware and software implementation. This is what this course has been trying to deliver. It is interesting to see how hardware and software are combined in a real-life application, especially with advanced technology such as machine learning and computer vision.

Unfortunately, because of the time constraint of the quarter system, we couldn't manage to connect all the pieces of this project. We have a fully working AI chess engine that gives choices for the best moves, but because of the small inaccuracy of the chess board detection, we were not able to connect those two pieces. The chess engine checks the validity of the moves, if the pieces are not correct even from the beginning of the setup, the game will not progress. Given more time, we can train our model better by enriching our data set and consequently achieving a better accuracy.

References

[1] Anton Elmiger. Jan. 2022. URL: https://github.com/aelmiger/chessboard2fen.

[2] Ashish Kumar Gupta et al. "Salient object detection techniques in Computer Vision—A survey". In: *Entropy* 22.10 (2020), p. 1174. DOI: 10.3390/e22101174.

[3] lhehontra. Dec. 2020. URL: https://github.com/lhelontra/tensorflow-on-arm.

[4] Szymon Wasik Maciej A. Czyzewskia Artur Laskowski. *Chessboard and chess piece recognition with the support of neural networks*. 2018.

[5] David Mallasén Quintana, Alberto Antonio Del Barrio García, and Manuel Prieto Matías. "LiveChess2FEN: A Framework for Classifying Chess Pieces Based on CNNs". In: *arXiv:2012.06858 [cs]* (Dec. 2020). arXiv: 2012.06858. URL: http://arxiv.org/abs/2012.06858.

[6] Yousifnimat and Instructables. *Chess robot*. Oct. 2017. URL: https://www.instructables.com/Chess-Robot/.