1. Introduction

A blackjack playing robot is our embedded system project. It is a pick-and-place robot arm in SCARA [1] configuration, capable of manipulating poker chips. Two stepper motors actuate the motion in-plane to the table, and servo motors actuate the prismatic joint and chip gripping end effector, achieving 3 degrees of freedom motion. A distance sensor allows the arm to pick up a desired number of chips. The mechanical components are 3D printed and mounted on a simple wood structure which is attached to the edge of a table. The Teensy 4.1[2] microcontroller controls the system.

The video demonstration can be found here: link

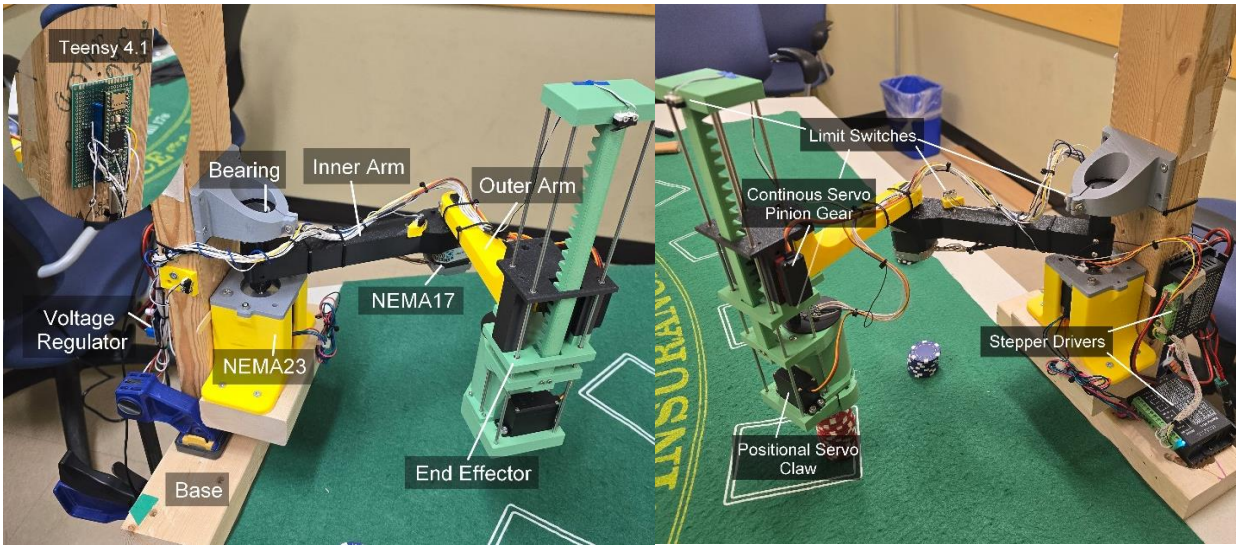


Figure 1 - System Components

2. Summary of the Project Proposal

The initial project sought to create a blackjack playing robot through five components:

1. Computer vision – detect and identify cards, assisting card counting, and chip detection.
2. Decision making algorithm – count cards and decide the play (hit, stand, double, etc.).
3. SCARA robotic manipulator – move in 3D space to bet chips and collect winnings.
4. End effector – grab chips and gesture the play. Takes an open-bottom cylindrical form.
5. Power systems – supply sufficient power to each subsystem.

The chosen microprocessor to handle the computer vision and deep learning was the Luckfox Pico Max [3] using the SC3336 camera module [4]. To handle the robotic arm manipulation, we decided to use the Teensy 4.1.

3. Alterations to the Proposal

The SCARA robotics manipulator, end effector, and power system components did not change greatly in overall function and design, while the computer vision and decision-making algorithm was altered in scoping for the project.

3.1 – Removing Computer Vision and Luckfox Pico Max

Due to unpromising camera performance in segregated tests and unexpected compatibility issues with the Luckfox Pico Max, our team decided to eliminate the computer vision aspect of the project, opting for a greater focus on chip manipulation than optimal gameplay.

Initially, the Luckfox Pico Max was chosen for its Rockchip NPU [3] which has strong computer vision computation capabilities, but the niche cross compiling process required to create a compatible YOLO model [5] caused versioning compatibility issues with the desired card detection model. Another issue was that the resolution of the camera module, SC3336, was not high enough to consistently detect/identify the cards. The camera must be placed sufficiently above the table to make all cards visible, which reduces resolution per card. Preliminary tests with images captured using the camera and tested with various online YOLO models yielded unpromising results.

Considered solutions were either using a better camera or having the camera move dynamically to identify cards, but we decided to instead scope away from this direction, focusing more on poker chip manipulation than playing blackjack. If computer vision worked successfully, the Luckfox Pico would have communicated the played cards over serial UART using the Rx and Tx pins of the Luckfox and the Teensy. Both devices use 3.3V, thus no level shifter would have been required.

3.2 – Sensor for Closed Loop Control

The initial plan for the vertical motion was an open loop control scheme, where the speed of the continuous servo is set, and distance is known by integrating (multiplying) the output speed by time of operation. This proved inadequate as the friction and mass of the system made it such that the continuous servo could not consistently move at the desired speed. Position could not be accurately estimated through an open loop system; thus, some sensor feedback was required.

3.2a – Rotary Encoder

The KY-040 rotary encoder was initially added to track the height of the end effector. This was done by attaching the encoder to the pinion gear, whose rotations can be translated to vertical motion, related by the geometry of the gear and the rack. Unfortunately, the resolution of the KY-040 was low, at only 20 pulses per revolution. This translates to 3.77mm per step. The height of a poker chip is about 3.45mm, which means we have insufficient resolution.

3.2b – Time of Flight Distance Sensor - VL53L5CX

Due to the insufficient resolution of the rotary encoder, a time-of-flight distance sensor was added to measure downward from the top of the chip cylinder. This change gives us several added benefits over the rotary encoder. Firstly, it provides an absolute measurement whereas the rotary encoder requires a known zero or initial position. Having a distance measurement also allows us to pick up a desired number of chips from a chip stack of unknown height. The distance sensor is a more effective sensor for using a control loop as it has a higher resolution and more meaningful measurements when sampled at a higher frequency. More details on the PID control loop can be found later.

4. Evaluation of Hardware

4.1 - Teensy 4.1 Microcontroller

This microcontroller selection is well justified given that the Teensy 4.1 effectively meets the requirements of our project. The Teensy 4.1 has numerous PWM channels that allow us to simultaneously control multiple motors. This allowed for the control of 2 servo motors, which were necessary for chip manipulation. Further, all digital pins support interrupts. This feature was essential for integrating interrupt-triggered limit switches which ensure that the end effector moves within the vertical limits. The I2C peripheral allowed us to easily connect the distance sensor. The Teensy 4.1 also has a lot of documentation which makes it easier to work with. The Teensy 4.1 has an FPU, which performs 32-bit float and 64-bit double math, a feature which we used to compute the inverse kinematics calculations.

The Teensy 4.1 has plenty of features that are unused in our project. These features include SD card integration, USB Host, SPI, audio, and many more. Most of these features are unused because we didn't have a need for it in this specific project. The SD card integration feature could be used to access audio files. The Teensy has 2 I2S pins, both of which could be used for stereo output. These features could have been used to implement audio to communicate the play of choice to assist the physical gesturing done by the robotic manipulator.

4.2 - NEMA Stepper Motors and TB6600 Stepper Motor Driver

To achieve movement of the SCARA manipulator in the XY plane, a NEMA-23 stepper motor actuates the inner arm, and a NEMA-17 stepper motor actuates the outer arm. The larger stepper motor is mounted at the base to provide a higher torque, since it must move the most amount of mass. The smaller stepper motor is chosen to actuate the outer arm as it is lighter, decreasing both the rotational inertia of the arm, and decreasing the moment caused by the weight of the motor. Both stepper motors are controlled with the TB6600 stepper motor driver, which has DIP switches to set micro-stepping amounts and current limits. Although the TB6600 was chosen for its superior convenience, the DRV8825 stepper driver can serve as a cheaper and smaller functional replacement.

4.4 - Servo Motors

A continuous servo motor (FT90MR) was initially used to move the pinion along the rack, achieving vertical movement. During testing, however, the combination of low torque and heat from prolonged use caused the motor to become faulty, occasionally stalling internally[1]. A stronger continuous servo (TD-8135MG) replacement was used, making the arm resilient to jamming. A mechanical claw at the bottom of the end effector grips onto the chips and is actuated by a positional servo motor (MG996R). This servo motor provides a strong holding torque.

4.5 – Micro Limit Switches

Four small limit switches are used in the final design. The inner and outer arms each have a limit switch detecting its home position, and the prismatic joint has a limit switch on each end of the

[1] We suspect that the plastic gears inside the motor became damaged due to heat and loading, as the motor later occasionally failed to spin when under no load.

rack. All limit switches have their normally open terminal connected to ground, and their closed terminal connected to distinct pins on the Teensy set to internal pull-up pins.

5. Evaluation of Software System

Initially, Arduino IDE was used to code the joint control for stepper motors. Its language is based on C and C++. To interface with the distance sensor, the vendor-supplied SparkFun library[2][6] was utilized, which handles all the low-level communication. Notably, the sensor requires its large firmware to be flashed over I2C every time on bootup, which would be tedious to implement ourselves. We also use the Wire.h library for low-level I2C implementation, which is provided by PJRC, the developer of Teensy 4.1. Additionally, the PJRC-provided PWMServo library helps to control servo motors using Pulse Width Modulation (PWM) in a non-blocking manner.

During integration of inverse kinematics and servo control programs, the code became lengthy and difficult to manage. Hence, the code was moved to PlatformIO, which assists in code organization given it properly supports linking multiple files. Moreover, PlatformIO supports IDEs such as VS Code, making library management simpler by allowing code to be written in separate files.

The distance sensor in use is the VL53L5CX, which is an 8x8 Time of Flight sensor by STMicroElectronics. The distinguishing feature of this sensor is the small size and the 8x8 grid, allowing one to generate a rudimentary pointcloud. However, since we only require one distance—from the top of the gripper assembly to the highest chip in the stack—we first down sample the sensor to 4x4 for higher frequency. (60Hz vs 15Hz, which makes a big difference in the PID control loop.) Then, we take the mean of the center 2x2 grid as the distance. Being an ad-hoc addition to the project, it is worth mentioning that we likely would have been better served by the VL53L0X (a single-distance ToF sensor), which effectively gives us equivalent functionality while being an order of magnitude less expensive. The VL53L5CX connects to the Teensy via the I2C protocol. For maximum throughput, we modified the I2C clock to be 1MHz while verifying the signal integrity through an oscilloscope.

6. Evaluation of Solution Approach

Due to the time constraints of the project, much of the algorithm was implemented with floating point arithmetic. Despite the floating-point unit on the Teensy 4.1, we would still likely benefit from implementing the complex math in the fixed-point format. This would allow us to perform computations faster.

Furthermore, the optimization techniques taught in class could be applied to much of the computations. For instance, we applied common sub-expression elimination in inverse kinematics calculations, which can be further applied to other parts of the code.

Moreover, lab 8 taught us the overhead of function calls, as we were able to improve the code execution speed at 30% simply by removing a couple function calls and directly performing actions in main. Although this would likely be much less noticeable on the Teensy 4.1, it would

[2] https://github.com/sparkfun/SparkFun_VL53L5CX_Arduino_Library Used in accordance with the MIT License

be an interesting experiment to use more macro functions, especially since we have plenty of program memory available.

In terms of program structure, our design could improve if we had spent more time integrating various components of the software system. The programs for each component of the arm were written separately and then later integrated once the arm was fully assembled. This proved to be inefficient given much of the code was not compatible with each other. This resulted plenty of code revisions. While the code was not optimized, the system works without any issues.

7. Overview of the Final Design

7.1 – Mechanical System

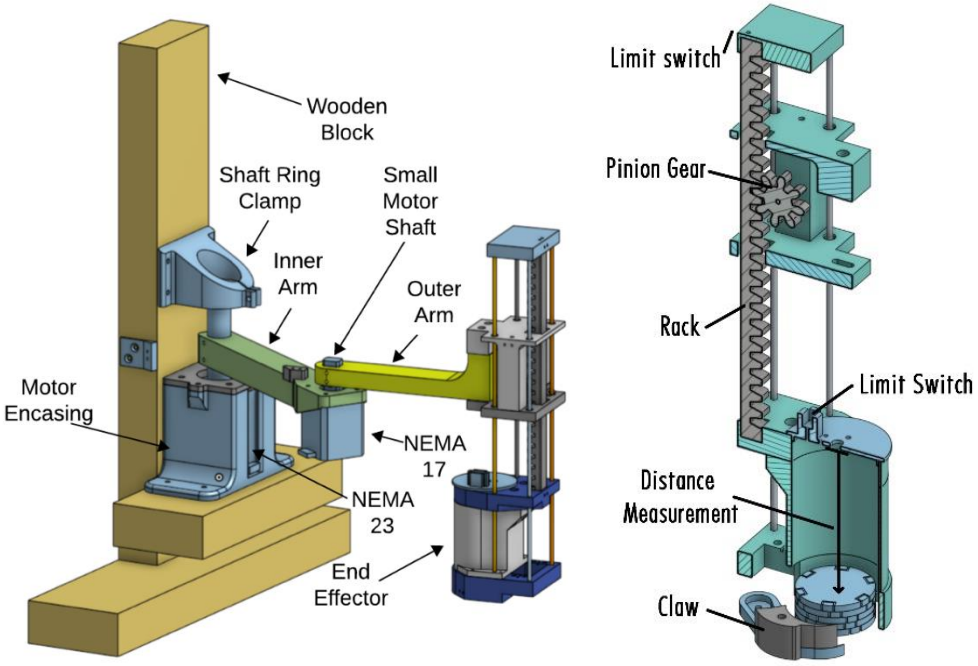


Figure 2 - CAD Assembly of the mechanical system

The NEMA-23 stepper motor is encased in a 3D printed casing. The enclosure is securely mounted on a wooden block, serving as a structural support. The NEMA-23 is wired to a TB6600 motor controller, which is wired to the Teensy 4.1. The NEMA-23 controls the angle at which the inner arm is positioned. The NEMA-17 is positioned at the end of the inner arm to control the outer arm. The inner arm was printed in ABS to withstand the heat generated by the motor during prolonged use. The NEMA-17 is wired to a separate TB6600, which is wired to the Teensy 4.1. At the end of the outer arm is the end effector. The end effector consists of the servo mounts, rack and pinion, and the grip. The continuous servo mount is attached to the inner arm, encloses the TD-8135MG motor, and is attached to the pinion. This servo is directly wired to a PWM pin on the Teensy 4.1. The TD-8135MG powers the pinion to roll along the rack causing the grip, which is attached to the bottom of the rack, to move up and down. The gripper contains another servo enclosure for the positional servo motor, which controls the position of the grip. All three parts of the end effector are connected through linear bearings, rods, and bolts where needed.

7.2 – Electrical System

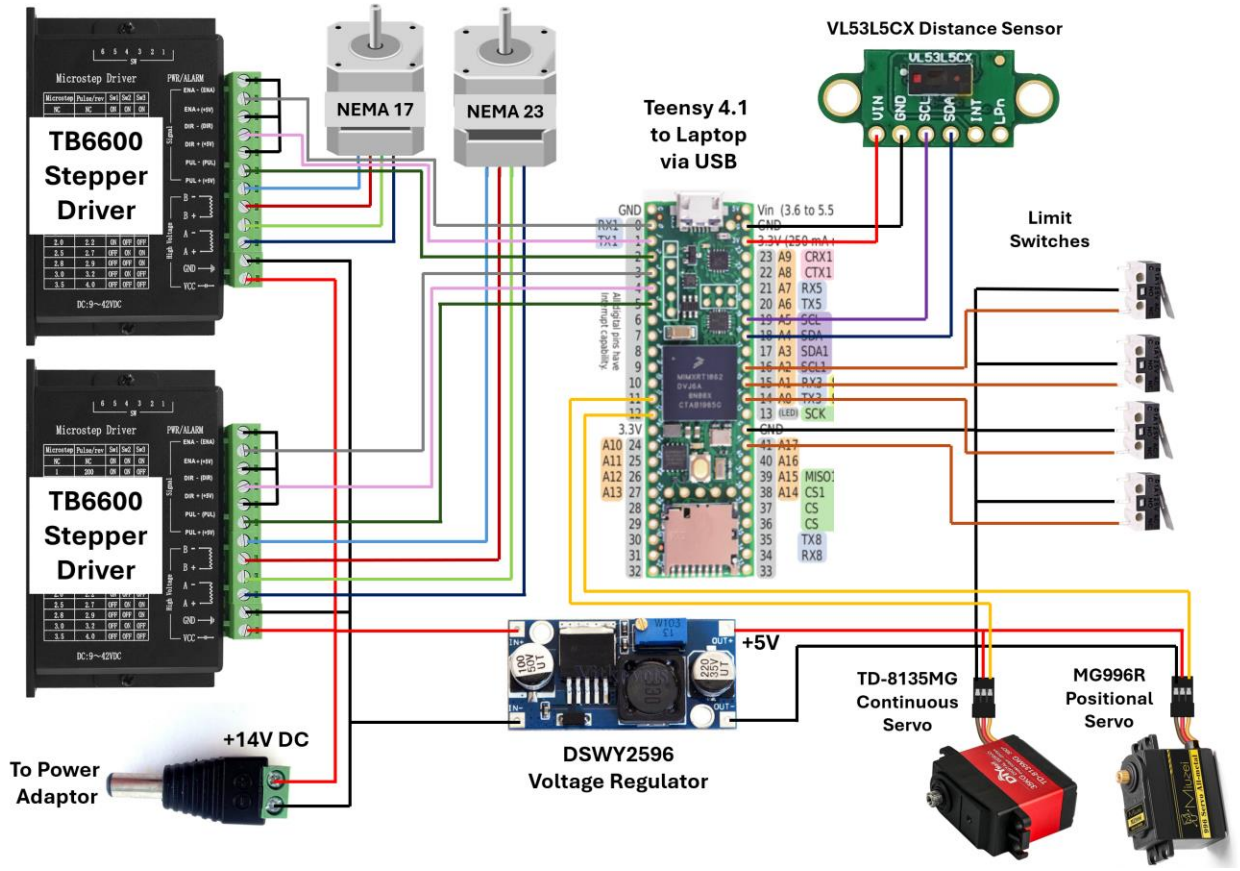


Figure 3 - Wiring Diagram of Robotic Manipulator

The system draws power from a variable power adaptor set to 14V DC. 14V is delivered directly to the TB6600 stepper drivers and the DSWY2596 voltage regulator. The voltage regulator uses an LM2596 step down regulator to output 5V to the servo motors (and previously the encoder). The distance sensor uses 3.3V powered by the Teensy, which is powered via a USB connection to a laptop.

The TB6600 stepper drivers' signal pins for direction and step are both wired to the Teensy's digital pins. The signal pins of both the TD-8135MG and the MG996R are wired to the Teensy's digital pins. The distance sensor SDA and SCL pins are connected to I2C_0 pins on the Teensy. The limit switches are wired to 4 interrupt-enabled digital pins on the Teensy.

7.3 – Software System

The whole system is running on a state machine that defines states like idle, moving to target, lowering, raising, etc. This helps maintains structure within the code, ensuring that the program is executed in the correct order.

Inverse kinematics (IK) was implemented to allow for easy manipulation. It allows the user to input end effector positions. The algorithm will calculate the individual joint angle to move the

arm. Invalid inputs are handled by early returning followed by a request for new input if the calculations resulted in ‘NaN’ or an unreachable joint angle.

To keep the stepper motors from stalling and skipping while moving, an acceleration algorithm was implemented. Once the arm receives the target joint angle, it starts moving toward that position by gradually speeding up. The system monitors its own speed and begins to slow down as it approaches the target position. The slowdown point is mathematically calculated such that, by using maximum deceleration, the arm smoothly stops at the target position. The steppers have been programmed for bidirectional movement, allowing the system to reach the target position faster.

As soon as the code starts running, the manipulator goes to a known home position, the absolute starting point. All other joint angles are relative to the origin, which is determined when the manipulator homes. This is visually depicted in the Appendix. Limit switches are used to trigger an interrupt as soon as the arm reached the known position.

Most of the code was written to be non-blocking for smooth multi-tasking, preventing freezing situations and better control. To make the serial input non-blocking, a function takes in serial input character by character, appends that character into a buffer. The serial input is prefixed by ‘<’ and suffixed by ‘>’. The suffix character terminates the string and returns the full buffer to let the rest of the code know there is a full command that was just sent. The stepper motor code has static variables which store the last timestamp when the pulse pin was last toggled. As soon as the time increases a set limit, the pin is toggled, and the motor moves one micro step. The entire movement code, including acceleration control, is non-blocking to facilitate a responsive user-experience. The limit switches are also implemented using an interrupt system which raises a flag when the ISR is called. Only certain parts of the code were written in a blocking manner by design. For instance, the homing procedure should be blocking to reduce software errors that could cause the arm to move without homing.

A PID controller is implemented on the end effector’s vertical motion. This helps to accurately reach the goal position (number of chips to pick) while reducing the steady state error to be negligible. We determine whether we have hit negligible steady state error by keeping track of the last time the error was greater than a small threshold value. If we have been under the error threshold for the last 1s, then we deem target to have been reached.

The input is taken through a usb-serial connection to a laptop. There are several available commands that can be given as the input, ‘mv, ‘stand’, ‘hit’, ‘gripper’, and ‘joint_mv’. ‘mv’ takes 5 operands: x_s y_s num_chips x_dest y_dest where x_s, y_s is the position of source chips (pickup position) in meters, num_chips is the number of chips to pick up and x_dest, y_dest is the drop-off position in meters. The coordinate frame is defined in the diagram in appendix. ‘stand’ makes the arm shake right and left by 40 degrees twice to indicate its intentions to the dealer. ‘hit’ makes the arm tap the table twice. ‘gripper’ takes one operand, if it’s 0, the gripper opens, if it’s 1, gripper closes. ‘joint_mv’ takes 2 operands, the destination joint angles in degrees to move the two joints, this doesn’t implement IK. ‘mv’ is the command that implements most of the software stack and is most frequently used. ‘joint_mv’ and ‘gripper’, as well as other miscellaneous commands are solely used for debugging purposes.

Microstepping is implemented in both the joints for smoother motion with higher precision. The motor approximately covers 0.05625 degrees per step. It also operates more quietly and reduces

vibrations. The acceleration with micro stepping puts much less mechanical stress on the system and prevents stalling.

7.4 – System Overview

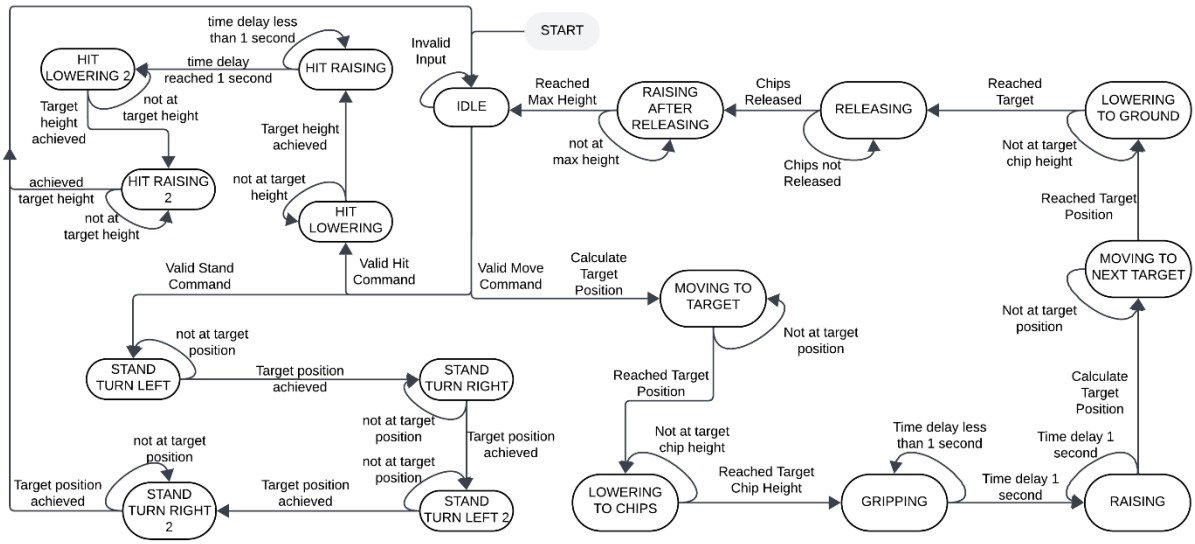


Figure 4 - State Machine Diagram of Jack the Gripper

The overall system integrates multiple subsystems, each of which are coordinated through the Teensy 4.1. This section outlines the overall system architecture, describes the hardware and software components involved, and explains how they interact to achieve gameplay. The robot functions based on a state machine, which is visually outlined in Figure 4. Initially, the bot is in an idle state, during which it waits for the user's input via serial. There are 3 valid inputs from the user: move command, hit command, or stand command. Upon the receival of an invalid input, the robot will remain in its idle state, prompting the user to submit a valid input.

Upon receiving a valid move command input, the robotic manipulator determines its current position through inverse kinematics and computes the necessary joint movements required to reach the target position. The bot then uses the stepper motors to achieve the correct target position in the workspace plane by manipulating the angles at which they are positioned at. The end effector then uses the distance sensor to measure the distance between the bottom of the end effector and the top of the chip pile. Using the PID loop described earlier, the servo motor moves the end effector to the correct height to pick up the right number of chips. The end effector makes use of the grip, controlled by the servo motor, to securely hold the chips. The end effector is raised up, with the chips in the grip. The next target location is then recalculated using inverse kinematics. The robotic arm uses the stepper motors to move the end effector to the second target position on the workspace plane. Once the second target position is achieved, the robot moves its end effector down and releases the chips. Finally, the bot raises the end effector such that when the state returns to idle, it can reposition the end effector without contacting any chip piles.

In the case that the user input is a valid hit command, the robot manipulator will then imitate the gesture indicating hit by lowering to a target height parameter. This parameter is tuned such that the end effector touches the surface of the table. After the end effector touches the surface of the

table, it then raises itself by running the continuous servo motor controller for a time delay of 1 second. Next, the servo will again lower itself to the target height, after which it will raise itself to its original position. The robot will then return to idle mode.

Finally, in the case that the user input is a valid stand command, the robotic manipulator will rotate the inner arm by 40 degrees, specified in the parameters. The manipulator will rotate this amount left, then rotate this amount right. This action is repeated twice before returning to its idle mode.

8. Testing and Tuning

Before the complete assembly of the entire arm, the individual components were tested.

8.1 – Inner Arm Testing

The angle of the inner arm when at the home position is known by design. Minor discrepancies in construction are accounted for by measuring the angle discrepancy of the arm when set to 90°.

The inner arm was tested by moving it to different angles and testing the accuracy to find out how much precision was repeatable. This helped to identify the critical need for stepper motor acceleration.

Moreover, it was verified in the earlier stages that the arm can hold and move the weight of the robot and the chips without stalling with the motors.

8.2 – Outer Arm Testing

Prolonged use of the NEMA 17 motor caused the motor to heat up. The current limit was set to the lowest setting to reduce heating. The heat caused PLA parts to lose rigidity, causing additional sag, so the shaft adaptor and inner arm were reprinted in ABS to help resist temperature failures.

Initial testing revealed that the motor stalled when run at a high velocity without acceleration. That amount of rapid acceleration is not needed for our needs, and acceleration for motors was implemented.

Moreover, initial testing revealed that there was a sag due to the weight of the end effector which had to be rectified in future iterations.

8.3 – End Effector Testing/Tuning

A critical test was measuring the continuous servo motor's capability in consistently moving vertically. Assuming the servo motor would turn at constant rate, the relationship of distance traveled to time actuated was determined by trial-and-error: if the servo motor incorrectly measured the height, then the code would be tuned, and the process would repeat. The servo motor was too weak to turn consistently, largely affected by mass and friction. Unsatisfactory performance of timed operation led to the inclusion of the distance sensor to measure the height of the end effector.

A rotary encoder was implemented but later removed due to its low resolution. The tuning value was calculated as follows:

$$\frac{dist}{step} = \frac{full\ rotation}{20\ steps} \cdot \frac{8\ teeth}{full\ rotation} \cdot \frac{9.425mm}{tooth} = \frac{3.77mm}{step}$$

The poor 20 steps per rotation resolution of the encoder led to linear resolution of less than the thickness of a poker chip (3.45mm). A time-of-flight distance sensor was used instead. To calibrate, the distance was measured with the gripper on the ground and empty. This was measured to be 115mm. Then the target distance for picking up n chips is as follows:

$$target\ distance = 115mm - n * (thickness\ of\ chip)$$

The claw was tested to see how strongly it could hold onto a chip. Similarly to the continuous servo, the ideal target angle to best grab chips was found by trial and error. The grip strength proved to be strong enough to lift 20 chips, which was deemed sufficient in our plans.

Integrated testing tested the effectiveness of both servo motors: to move the end effector up and down with the additional weight, and to firmly grab onto chips with vertical motion. During testing, it was found that the claw would sometimes fail to pick up the bottommost chip. This was likely due to sag in the system varying across different positions. This was later improved by tightening the shaft-arm connection.

8.4 – Limit Switch Testing

Each limit switch was wired up to ground and a pin set to pull up mode. The switches were tested by setting each pin to interrupt on the falling edge, then pressing the limit switch. An interesting bug was found, in which the limit switch connected pin 13 caused unexpected interrupt triggering. It is suspected that the internal LED connection on pin 13 lowered the voltage of the pin enough to cause unwanted triggering, as the LED draws some current when the pin is set high.

9. Future Features

Beyond improvements that could improve the function of the system, there are plenty of directions this project could develop into. Implementation of camera vision as initially intended would give a level of autonomy to the system, allowing it to make decisions by itself. Different games could be programmed into the arm, so it could play other games that require chip manipulation. Further precision improvements could possibly allow it to stack chips on top of existing stacks. Path planning could be implemented to allow motions that avoid obstacles or move in complicated manners.

10. References

[1] C. to, “type of industrial robotic arm,” *Wikipedia.org*, May 28, 2006. https://en.wikipedia.org/wiki/SCARA (accessed April. 7, 2025).

[2] Adafruit Industries, “PJRC Teensy 4.1 Development Board,” *Adafruit.com*, 2025. https://www.adafruit.com/product/4622 (accessed April. 7, 2025).

[3] “LuckFox Pico Max RV1106 Linux Development Board,” *RobotShop Canada*, 2023. https://ca.robotshop.com/products/luckfox-pico-max-rv1106-linux-development-board (accessed April. 7, 2025).

[4] “CSI Camera | LUCKFOX WIKI,” *Luckfox.com*, 2024. https://wiki.luckfox.com/Luckfox-Pico/CSI-Camera/ (accessed April 7, 2025).

[5] Ultralytics, “YOLOv8,” *Ultralytics.com*, 2023. https://docs.ultralytics.com/models/yolov8/ (accessed April 7, 2025).

[6] Sparkfun, “Sparkfun/sparkfun_vl53l5cx_arduino_library,” GitHub, https://github.com/sparkfun/SparkFun_VL53L5CX_Arduino_Library?tab=License-1-ov-file#readme (accessed Apr. 7, 2025).

11. Appendices

Workspace Coordinates

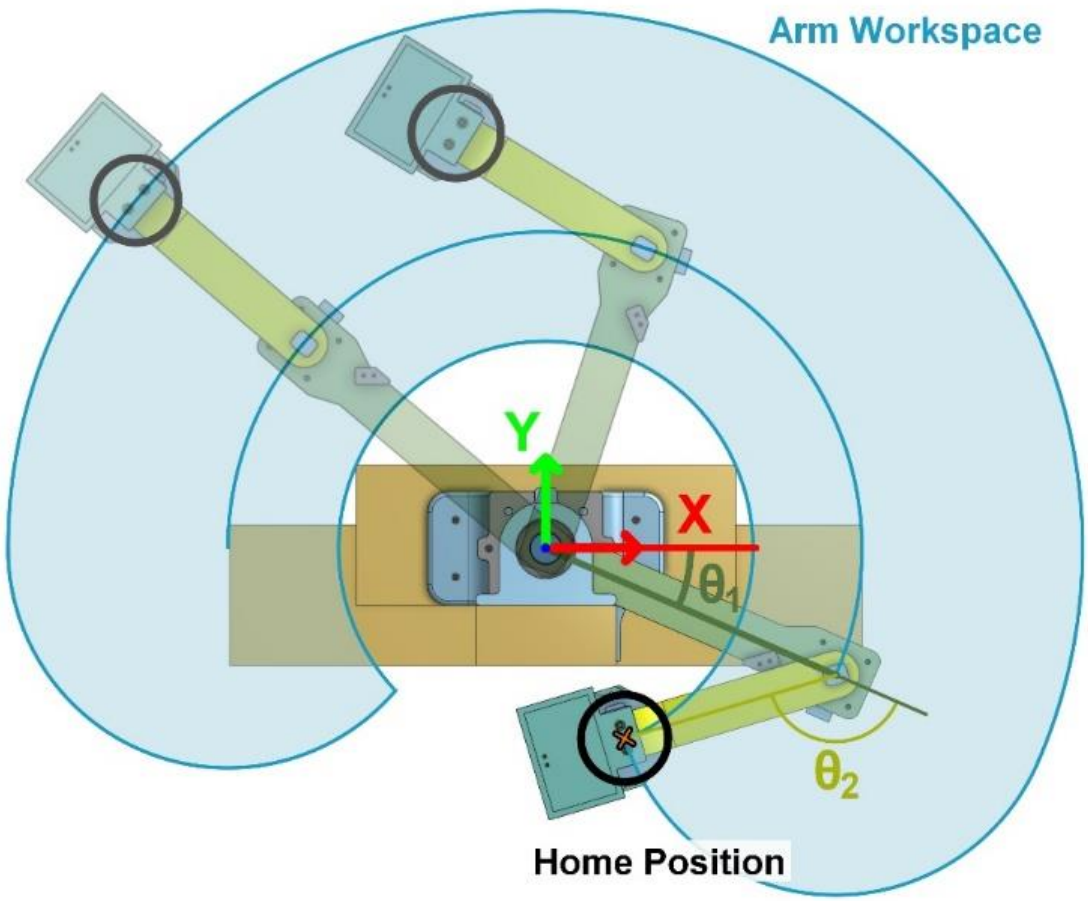


Figure 5 - Workspace of Robotic Manipulator

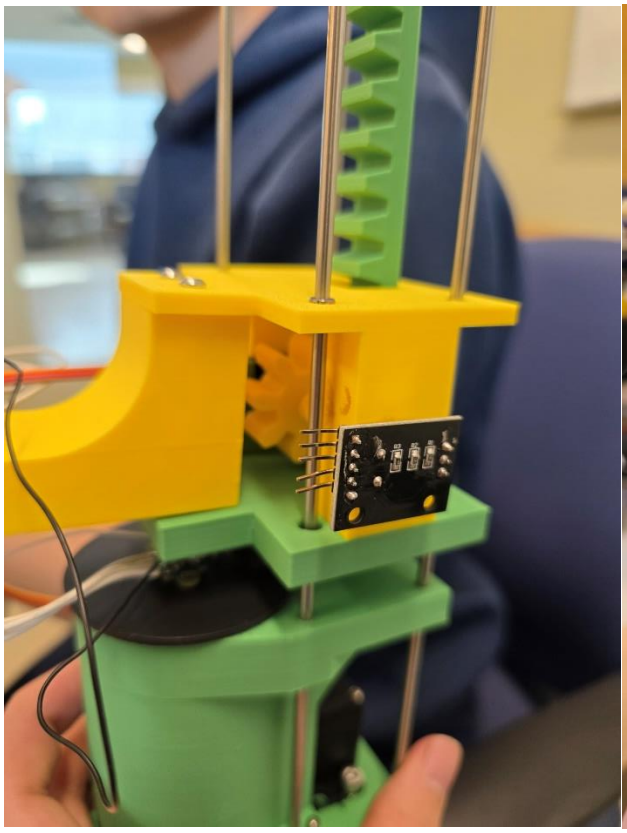
The workspace of the system is shown (blue region), with the coordinate variables labelled. The home position is shown as the bottom right configuration. The inner arm has angle range -33 to 180 degrees, whereas the outer arm has -140 to 140 degrees.

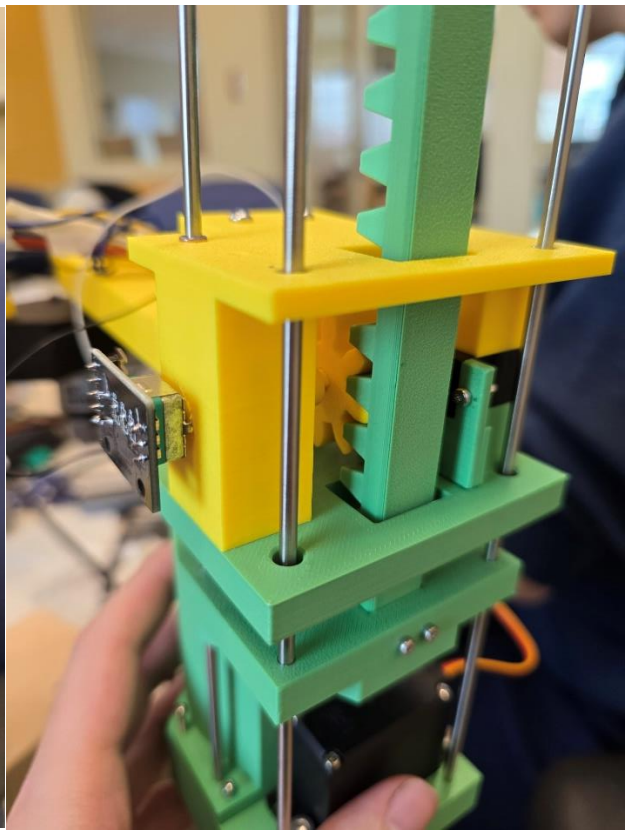
Example YOLO Model photos

The model mischaracterizes many of the cards.

Photos were taken on the Luckfox Pico Max SC3336 camera.

Rotary Encoder Design





The pinion gear has a shaft that extends out to the rotary encoder. The rotary encoder was later removed thus no longer be found on the final design. This version also uses a smaller servo motor, which was replaced with a larger servo motor in the final design.

The MIT License (MIT)