IEEE Southeastern Hardware Competition Senior Design Report

University of North Carolina Asheville and North Carolina State University

Joint Engineering Mechatronics Program (JEM) 485 Senior Design

1 May 2024

Co-Authored by: Carlos Anzola, Chris Bass, Alex Haines, Zachary Price, Jason Robinson, Maxwell Turcios

Faculty Advisor: Dr. Eli Buckner

Introduction:

In the realm of competitive engineering, innovation, and proficiency converge to redefine the boundaries of technological advancement. In this report, we present the culmination of our efforts in the 2024 IEEE Southeastern Conference Hardware Competition, an event hosted by the Institute of Electrical and Electronics Engineers (IEEE). Our task? To design and construct an autonomous robot capable of completing designated tasks within strict constraints.

The competition, open to every southeastern university boasting an engineering program, sets forth rigorous rules and limitations to challenge the ingenuity of participants. Our robot, like those of our peers, must operate autonomously, be no larger than 1ft x 1ft x 1ft upon starting the course, weigh under 25 pounds, and ensure no threat or damage to the playing field. These parameters not only test our technical prowess but also foster an environment conducive to creative problem-solving.

While the pursuit of victory fuels our competitive spirit, our motivations extend beyond mere accolades. Participation in such events allows us to maintain invaluable relationships within the IEEE community, showcase our engineering proficiency, and elevate the visibility of our respective institutions' mechatronics programs. Moreover, the exposure garnered from competitions of this caliber often translates into tangible job opportunities for aspiring engineers.

Guided by these aspirations, our team embarked on a strategic journey toward success. Adopting a phased approach, we began by developing a Minimum Viable Product (MVP) that satisfied the competition's basic requirements, before iteratively enhancing its capabilities. This iterative process was facilitated by the division of our team into specialized subgroups, each focused on distinct aspects of the project: mechanical design, electrical systems, state machine programming, robotic arm manipulation, and computer vision integration.

In the subsequent sections of this report, we delve into the intricate details of our robot's design, the challenges encountered along the way, and the innovative solutions devised to overcome them. Through meticulous planning, relentless dedication, and unwavering teamwork, we present a testament to the boundless potential of engineering ingenuity in the pursuit of excellence.

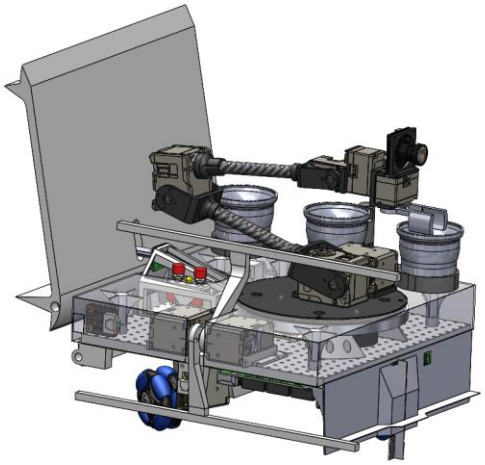
Main Mechanical System:

Mechanical Frame:

The prototyping and design for the robot was all done on a 3d printed protoboard. It was designed to be as large as the nice bamboo lab printers could produce, and just stiff enough to have little to no flex without being noticeable heavy. The holes spaced out in a 10mm grid were tested so that a M3 screw could be fastened to the board without any additional hardware. However, if needed the holes could be striped out and nuts placed on the other side for additional support. This frame served as the base for our cad model.

3D Model:

All parts that were added to the robot were first downloaded from the manufacturer website, or created manually in solidworks. These parts were then added to the frame, assembly, and positioned in the desired location. A new part was then derived from the existing model to perfectly fit the custom geometry of the part, and line up exactly with the M3 holes. Using this method parts could be solidly secured to the robot in as little as 30 minutes for rapid prototyping. Since these parts were screwed into place the quick prototyping of parts could also be used for the final solution. Building the complete bot in Solidworks before any stage of assembly was time consuming, but effective in the long run. There were little to no unforeseen complications mechanically with the insight of the full model inside a mock arena.

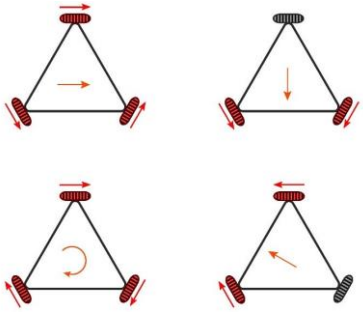


Simulations:

The 3D models were also used to simulate loading, and clearance for testing. The robotic arm brackets and tubing were subjected to finite element analysis inside solidworks in order to determine the flex of the arm under max loading. This allowed me to make decisions to maximize weight lifted with little compromise in accuracy. The robot was also moved around in the simulated board to test for things like clearance and arm reach. This allowed for drastic reduction in prototyping time with the ability to test wheel configurations and part clearances without the need for printed objects.

Wheels:

We used three omni-wheels in a triangle configuration for the movement of the bot. Omni-wheels roll like a normal wheel, but have small rollers on them that allow for perpendicular movement freely. Set up in this configuration, omni-wheels allow for full 3 degrees of freedom in a 2D plane. By controlling the speed of the individual wheels, the bot can move in the x, y, or rotate in the z independently and simultaneously of each other. The three wheel configuration means that even in uneven terrain, all three wheels will be contacting the ground, which is essential for proper movement.



Dynamixel Servos:

The wheels mentioned above, the robotic arm, and all other moving subsystems were all run with dynamixel servos. Dynamixel servos are a compact all in one servo that comes in many sizes that allow for control and reading of dozens of values inside each servo. The servos can be wired in series, making wire management much easier, and communication through a single pin. Each servo is assigned an id, and can be set to different modes such as velocity, position, extended position, PWM, and current control. The wheels on our bot were velocity controlled, the gripper was current controlled, and all other servos were position controlled. In addition to the mode, the max acceleration, max velocity, timing profiles, PIDs, position limits, and offsets were assigned and saved. During normal operation the servo goal position/velocity was set, and the current position/velocity was monitored. Any overheating, over torquing, or any other potentially damaging action would trip a safety and stop the harmful action. All of the control was handled though our servos ROS node using the Dynamixel library.

Robotic Arm:

Arm Specifications

In order to accomplish all the miscellaneous tasks in the competition, we agreed very early that a robotic arm would be a great option to pursue. Taking lessons from a previous version of a robotic arm, and valuable insight from my internship, I designed an arm to meet our demands. This arm would be a 6 degree of freedom arm, allowing for control of the translational xyz and rotational roll, pitch, and yaw of the end effector. It can reach up to 16 inches from the base, or about 14 with the gripper positioned down. The maximum payload is a half a pound at max reach, with a quarter pound as the safe working load. The base uses two servos in parallel to generate the torque needed for this. The servos used have built in encoders that provide less than a degree of accuracy for super precise positioning. The whole thing is programmed and controlled in ROS.

Arm Design

The arm was designed to maximize the length of the reach while shooting for a half pound of payload. Given these servos I calculated the position of the joints and arm mass to give me as much reach as possible. I then performed finite element analysis on the links to calculate the flex under heavy loading conditions. Although there was a few mm of play under heavy loading conditions, under the conditions we were planning to use, this was not an issue, and the extra length of the light arm made up for this shortcoming. The joints were designed in a way to allow for maximum flexibility, with three joints having 180 degrees, and three joints with full 360 degrees of movement. The joints were also designed in such a way that frame assignments, and future kinematics would be as simple as possible, which very much paid off. The joints were all designed to be easily 3d printed and replaceable.

Arm Fabrication

When it came to to build the arm, 6 dynamixel servos, several 3d printed brackets, a turntable bearing, and some half inch carbon fiber tubing was used. The carbon fiber tubing is pressed, and glued into the 3d printed brackets, which are screwed onto the servos. The large servos allow for a bushing to be placed on one of the sides, to allow for wires to be fed through nicely. This allowed for very good management of the 5 wires that run all the way up the arm. 12v, signal, ground, and a pair of spare rx/tx wires supply all the power and communication needed for the arm and camera at the end. The wires are braided, heat shrinked, and hot glued for a streamlined robust build. There is a voltage regulator at the small servos at the end to supply the 5 volts needed for the smaller servos and camera. These wires are then supplied power/ground, usb connection for the camera, and the signal wire is connected to the U2D2 that interfaces with a controller. In the case of our bot, a raspberry pi.

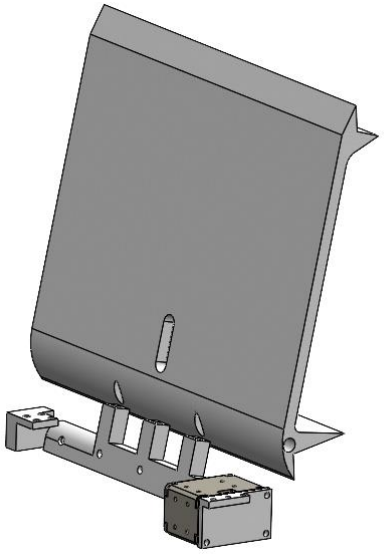
Arm Kinematics

The original tasks required for complex rotation of the boxes when delivering, this led to the 6 DOF arm in the beginning. I derived forward kinematic equations, and numerical inverse using matlab. However, the inverse kinematics was a very lengthy process, sometimes taking up to 10 seconds to find a solution. Since there are multiple solutions possible, the solution it gave wasn't always the most desirable. After simplification of the rules, we no longer needed complex rotation of the wrist. This allowed me to constrain the wrist to alway be pointed down, and joint 4 to be locked at a home position. With these conditions I derived the analytical inverse kinematics for 5 DOF to give only the solution we were interested in. This made for near instantaneous joint calculations, and flawless positioning that was implemented in ROS.

Other Mechanical Subsystems:

Bridge:

The bridge was originally our backup design for crossing the gap, but proved to be very consistent and reliable. It was designed so that once the bot had backed up to the gap, it could be lowered, caught on the edge, and fall into place. A single servo was used to lower the bridge. A small slot was cut into the floor to allow the led detector to show through. Extra rotating catches were added to increase the catching profile of the edge, without increasing the starting profile. Lastly the bridge was made as light as possible so as to not cause the bot to tip over backwards while going up slopes.



Jaws attachment:

The jaws attachment was a container designed to fit in the gripper of the robotic arm. Its sole purpose was to “eat” and hold all the small blocks for transport and drop off. The mouth of the gipper was made of 3D printed bristles that were developed to be stiff enough to allow capture of the small blocks with minimal effort, while not allowing the block to fall back through when picked up. The base was made to funnel the blocks toward the center, where 5 sets of bristles are stacked together for the best holding results. The entire container was printed with tolerances to be able to be taken apart easily for retrieval of the blocks. The container was held firmly by the gripper, but was allowed to wobble side to side and front to back for better tolerance when picking up blocks not quite directly underneath.

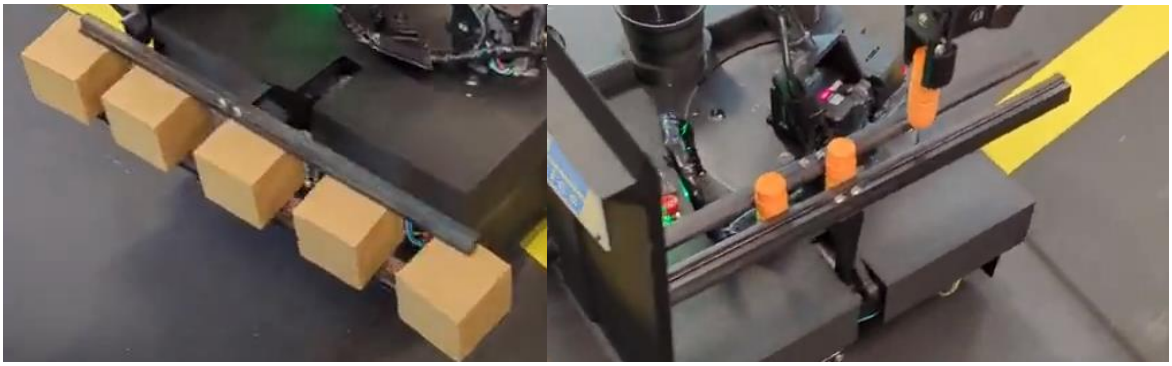
Cylinder holder:

The cylinder holder was designed to hold all three cylinders at the same time. It uses the same bristles designed for the jaws to allow the easy placement of the cylinders without falling out

when picked up. The top of the holder has funnels which guides the cylinders down into center, even if they are crooked. Once all of the cylinders have been placed inside the holder, and the bot has arrived at the destination, the arm picks the whole assembly up off the magnets holding it down, and places them all at once.

Bulk Grabber:

The bulk grabber was designed to pick up the large blocks, and small cylinders all at the same time. Since these items were always in the same location, the robot would drive to a precise location and pinch either side of the objects, then rotate upwards. This allowed us to save time since the robot arm can only pick up one item at a time. It also allowed the sorting of the cylinders while driving to the final destination. The long part of the arms themselves are made from carbon fiber rods to help prevent bending. Secured to the edge of the rods is a strip of silicon. This not only allowed for a grippier texture, but provided a bit of give to conform around the objects. This design allows for an item or two to be dropped without losing the whole lot. The servos for these systems are tuned without I term. If the I term was included in the PIDs, then the servos would work harder and harder till they tripped from overcurrent.



ROS Servos Node:

Robotic arm Control:

The original purpose of this node, and the bulk of it was dedicated to was the kinematics, commanding, and monitoring of the robotic arm. Inside this node the inverse kinematic would take the desired task space, and convert to a set of joint angles using analytical kinematics derived to only give a single solution for 5 degrees of freedom. Then arm angles could take this data or custom angles given directly from the user and prepare a list to be written to the servos. Once a new set of values is given the node will continuously check to see if the arm has arrived at the specified destination within a desired tolerance. The tolerance and well as the movement speed is specified in either the taskspace or angles topic. The arm also had the ability to perform linear interpolation if needed. The user would specify a starting taskspace and distance to travel, and the joints would all be linearly scaled to take a set amount of steps to the destination. These scaled velocities would mean the arm would not need to be checked at each step along the way, and could run quickly and smoothly till the final step which would check for the arrival. In addition to the commands given to the arm it is also capable of providing custom feedback to the user, usually in the form of current angle or torque upon request.

Misc Servo Control:

A relatively small section of the node is dedicated to the control and monitoring of the servos used in the bulk grabber, bridge, and flag. This would take in a value or set of values that represent joint angles of the servos. If some angles were not provided then old angles were used instead. Once a new angle or set of angles was declared then the node would monitor those servos until within a desired tolerance. For this monitoring I also had to add the ability to declare arrived if no progress was made for a set amount of time, since during the pinching moves of the bulk grabber, the servos would never arrive at the desired position.

Bot Movement:

This section of the node controls the accepting of new movement commands, wheel kinematics, movement PIDs, movement speed, monitoring arrival, and enabling/disabling sensor readings. The bot was moved around the board based on lidar and imu sensors passed into the servos node from the sensor node. This constant publishing of data was taxing on the system, so once the bot was at the location it needed to be, the sensor posting was told to be turned off. While the sensors are being posted though, the callback function that is triggered by it would call all of the xyz PIDs, and update the wheel velocities with the most recent readings. If there had been no readings, no new velocities were needed. There was also a built-in safety to pause wheel movement if no reading had been received within an allotted time, and resumed once reading began again. Another safety was added to stop the wheels if the rear TOF that was facing down read more than a set value. This was very helpful in keeping from running into the gap, and simply picking up the robot to get the wheels to stop. The PIDs could also be changed with the

posting of a topic, so there was no need to recompile and launch the node for tweaks. The tune was very tight, and put us at the desired location/rotation nearly as fast as possible with the current hardware, with minimal overshoot, and only a couple mm of repeatable error.

Extra functions:

Other than communicating with all the servos and performing other functions for the above systems this node also served a few misc purposes as well. All the servos were set up and initialized, this included passing in unsaveable values, and setting up sync writes. Sync write allows for pre-setting a list of things to write to servos, then outside of any callback functions, writing all those servos at once. This was particularly useful in streamlining ROS callbacks, and helped solve missed commands. Finally the node would look for any error messages every few seconds from the servos and reset those servos that might have faulted.

Electrical System:

Power Supply and Battery Management:

The robot uses a four-cell lithium-ion battery with a voltage ranging from 16.4 to 12.8 volts. Lithium-ion batteries have a high energy capacity but low weight. However, such batteries can be irreparably damaged if the voltage of each cell gets too low. Systems that draw too much power will also be damaged. The team decided to protect the robot by designing a power management circuit and circuit board. If below 12.8 volts, the power management circuit will disconnect the battery. But the circuit will also disconnect if the robot pulls more than 10 amps.

Power Regulation:

Each subsystem has its power requirements. The robot operates at one of two voltages: 12 or 5 volts. Buck converters provide these voltage levels. A heavy-duty buck converter steps the battery voltage down to 12 volts. Multiple smaller buck converters step the 12-volt converter down to 5 volts. Note that the sensors take power from the 5-volt converters on the Arduino Mega.

Sensor Node:

An Arduino Mega interfaces with sensors and publishes localization data to ROS. The robot utilizes I2C sensors, allowing every sensor to communicate through the same data and clock lines. A custom-designed circuit board manages the power, data, and clock lines. Additionally, the board comes with a power regulation IC for a potential 5-volt supply.

Computer Vision System:

What is Computer Vision:

Computer vision allows the robot to obtain information from the visual world.

Purpose of the Computer Vision System:

The purpose of a computer vision system in the robot is to identify the locations of purple blocks and fuel tanks with respect to the base of the arm.

Vision System Process:

Image acquisition:

The robot uses an Arducam 5MP wide-angle camera located at the top of the end effector to take a picture of a specific area.

Distortion Removal:

Since the camera has a wide field of view (FOV), the distortion on the image is considerable and reduces the accuracy of the posterior coordinate calculation.

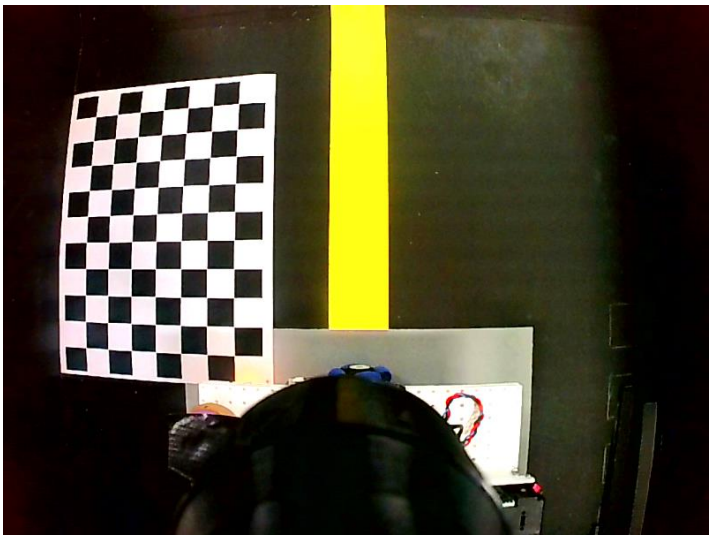
There are two primary types of distortion: radial distortion and tangential distortion.

- Radial distortion makes straight lines appear curved, with the distortion increasing as points move further from the image center.

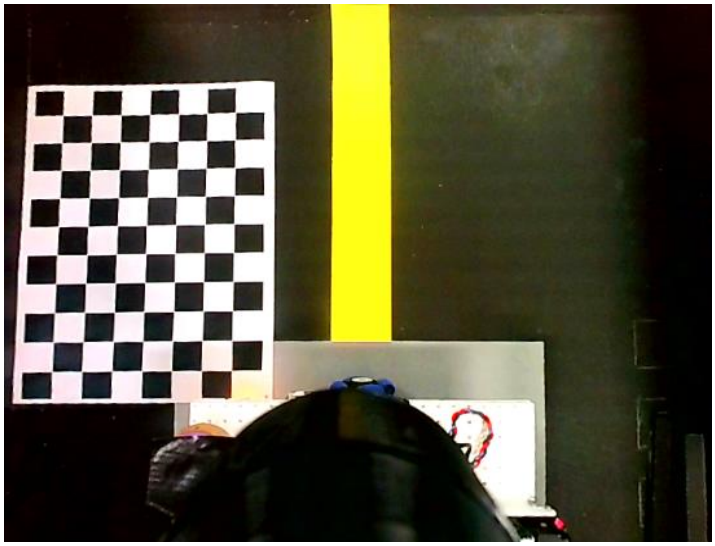
- Tangential distortion arises when the camera lens is not perfectly aligned parallel to the imaging plane, causing certain parts of the image to appear closer than anticipated.

To remove the radial distortion, the camera was calibrated using a checkerboard pattern. Camera calibration determines the parameters of a lens, image sensor, or video camera, which can be used to estimate structures and remove lens distortion.

To reduce tangential distortion, every time the robot needed to take a picture, the camera was placed in a position as close to parallel as possible with respect to the plane of interest.



Original Image



Undistorted Image

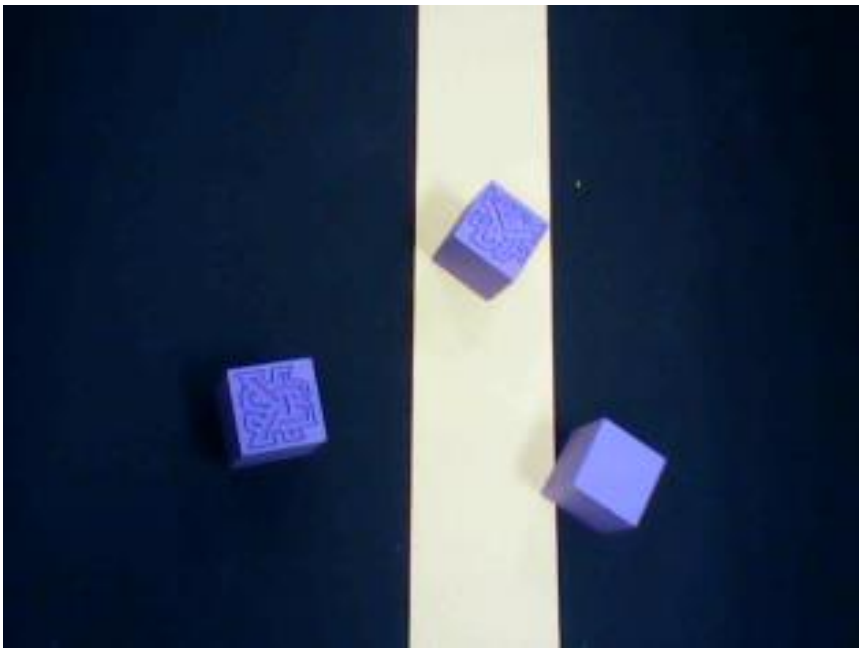
Color Filtering:

After obtaining the undistorted image, the robot filters the colors in the image to preserve only the objects of interest. Since every object has a specific unique solid color, color filtering allows the robot to remove anything that is not the object of interest.

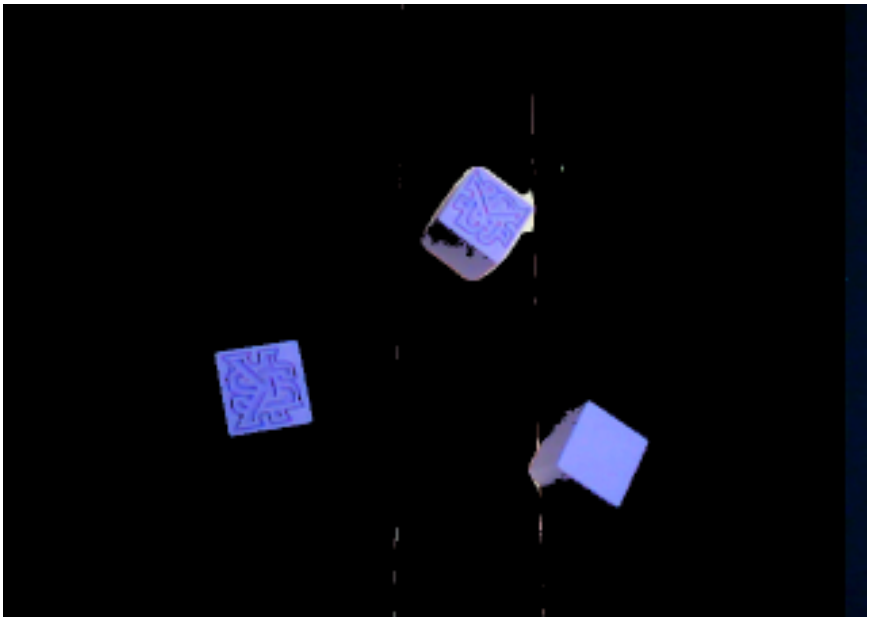
There are three important preprocessing steps to improve the color filter accuracy:

- Reducing the contrast in the image to balance highlights and shadows.
- Increase saturation to make the colors more vibrant and homogeneous.

- Converting the resulting image to hue-saturation-value colorspace (HSV) to have direct access to the channels controlling color, highlights, and shadows. This helps make the filter more flexible by accounting for different shades of the same color.



Original Image



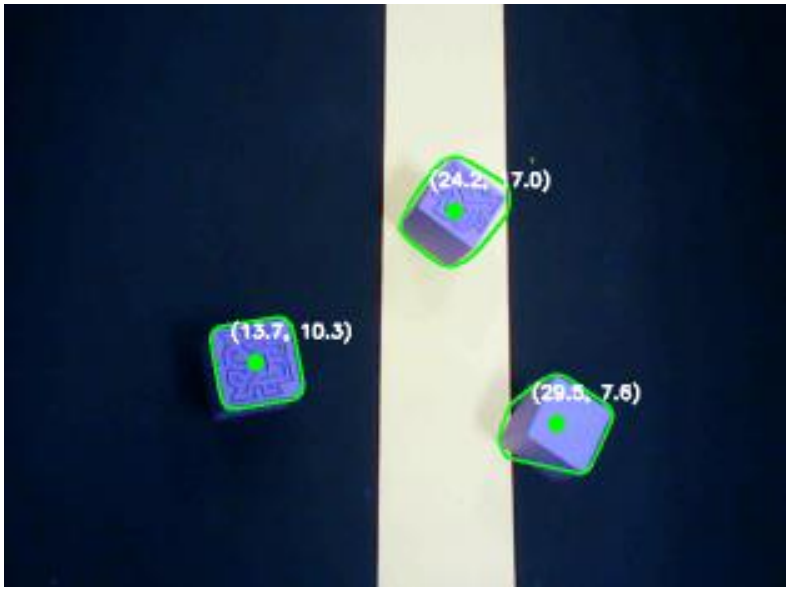
Filtered Image

Coordinate Calculation:

For each object detected after the color filtering stage, the robot can obtain different characteristics for that region. For coordinate calculation, the robot obtains the centroids of the regions detected by using the image moments.

Image moments represent weighted averages of pixel intensities or functions thereof, chosen for their meaningful interpretation. They serve as descriptors for objects post-segmentation, providing insights such as area, centroid, and orientation.

After the centroids are found, their pixel coordinates are converted to meters by using a conversion function.

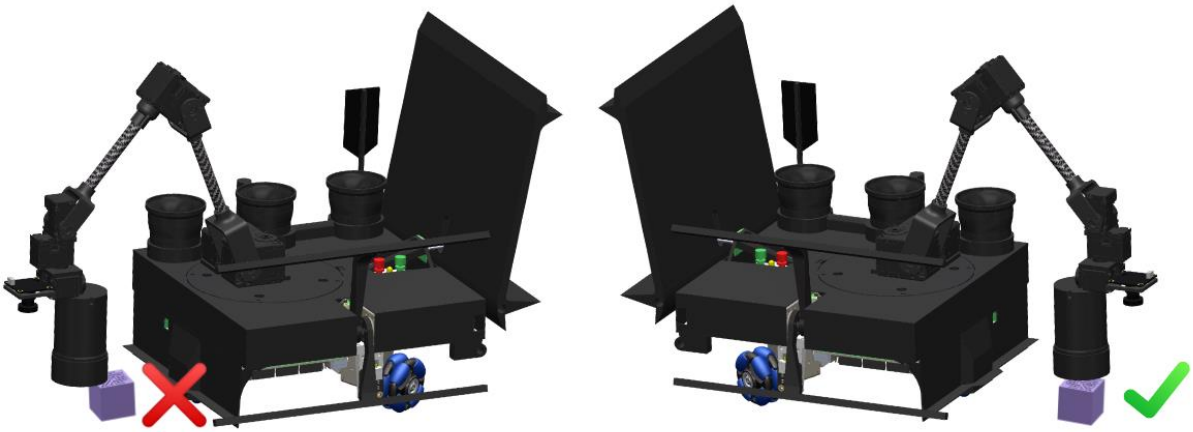


Coordinates Image

Error Correction:

Drawing from experimental data obtained through multiple attempts to pick up purple blocks and fuel tanks, an error model can be constructed. Polynomial regression is employed to derive this model.

Various random locations were chosen for the objects of interest, and the corresponding distance errors in successfully picking them were recorded. These data points were then utilized to formulate an equation characterizing the error along both the X and Z axes for each item.



Dramatization: No Error Correction VS Corrected Error for Purple Block Pickup

State Machine in ROS:

What is ROS:

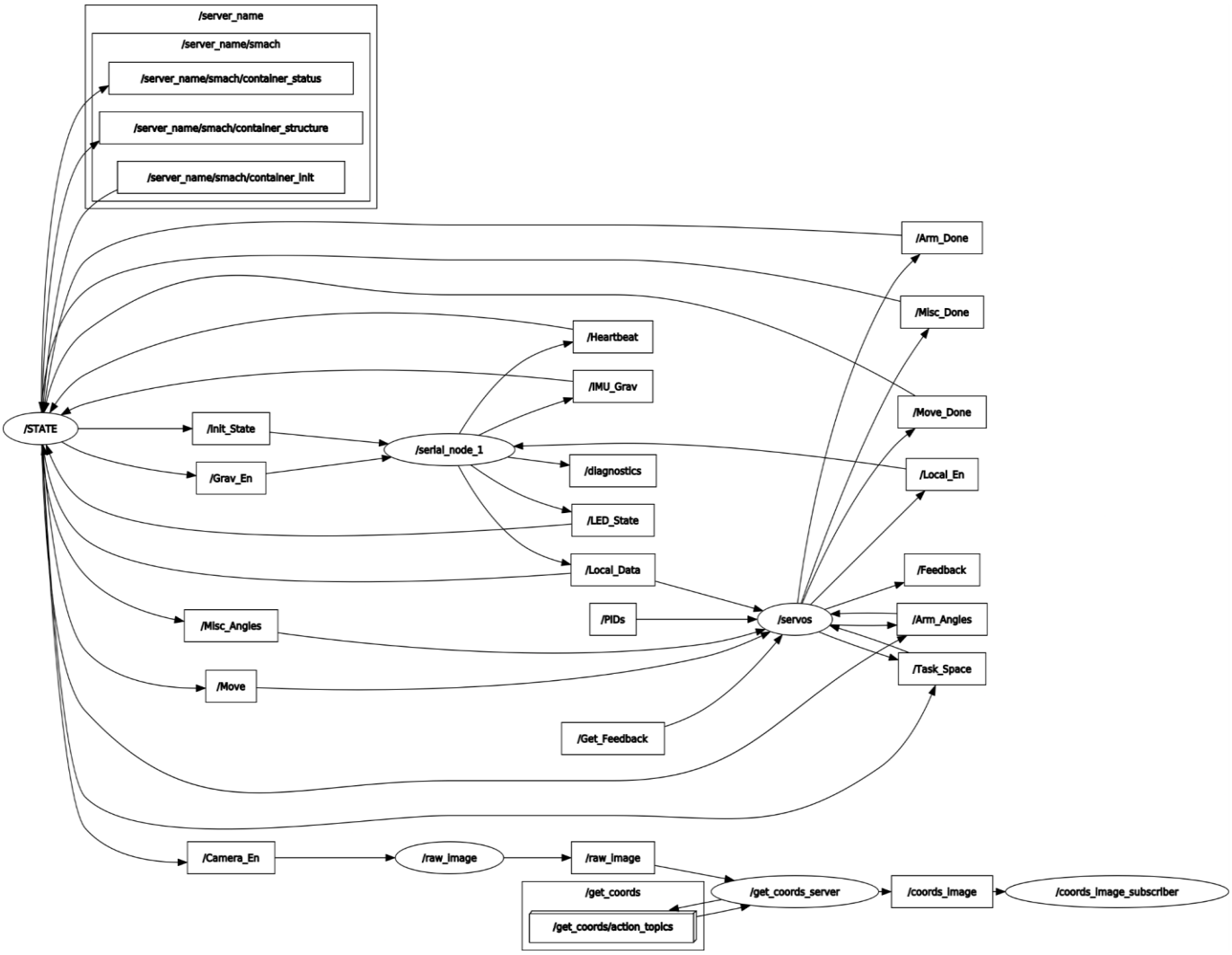
ROS stands for Robot Operating System. It's a framework for writing software that is open-source. Contrary to what its name suggests, ROS is a collection of software tools and libraries that assist programmers in creating robot applications rather than a conventional operating system. Hardware abstraction, device drivers, interprocess communication, package management, and other features are all provided by ROS. It is extensively utilized in both industry and research to create and manage robotic systems.

How does ROS work:

ROS uses a system of publisher and subscriber nodes in order to communicate with other programs. In other words, publishing can be also described as transmitting and subscribing as receiving information or messages. These nodes publish and subscribe to topics, which can be sensors, values of servos, etc. This allows for a flexible workspace since code in C++ and Python can communicate with each other. Arduino and Raspberry Pi are able to easily communicate with each other despite being two different microcontrollers.

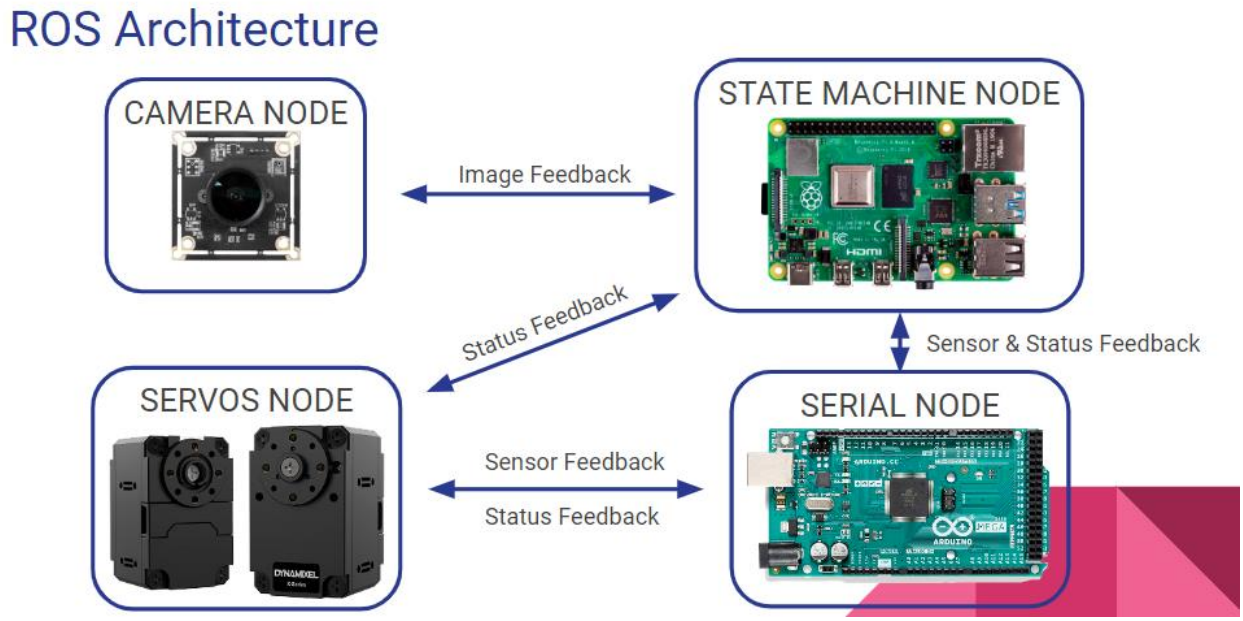
ROS in our State Machine:

The team chose to integrate ROS for its flexibility and robustness. Since there are two different microcontrollers, the application of ROS is acceptable. For ROS to work in the way that is required for this project, Ubuntu and ROS Noetic were downloaded on the Raspberry Pi. Moreover, the ROS server had to be set up to allow the publishing/subscribing of messages between nodes. After setting up, writing code with ROS libraries and tools was necessary for the framework to work. Below is a diagram of the relationship between nodes in our robot:



The state machine is a behavioral model that changes its status based on inputs. For example, the sensors in the robot scan the environment and that gives specific information that may change the status of the state machine. It uses ROS to communicate with Servos and Sensor Nodes. The Sensor Node has the TOF sensors and IMU, these are essential for the state machine. Many of the functions and states rely on the information given by the Sensor Node. The Servos Nodes is used for the movement of the wheels and the robotic arm. This is used to pick up purple boxes, brown boxes and orange fuel tanks. The structure controls the instructions given to the robot. The way that the state machine decides which state will be executed next relies on the status feedback from the camera node, servos node and sensor node. Below is a

picture of the Architecture:



Once this feedback is collected, a decision is made on whether it can move to the next state or not. Within the overall state machine, there are concurrent state machines. This means there are other state machines working in conjunction with the main one. This is used to make code easier to read and write.

Results:

In the 2024 IEEE Southeastern Conference Hardware Competition, our team demonstrated remarkable proficiency across multiple facets of the event. Our efforts were rewarded with a 1st place win in the Design Competition, a testament to the innovative and meticulously crafted design of our robot. In the Hardware Competition, we secured an impressive 9th place out of 54 competing teams, maintaining the 2nd highest total points throughout the competition. Notably, our performance was highlighted by achieving top scores of 104 and 107 points in the first and second rounds respectively, showcasing the effectiveness of our robot's capabilities. Furthermore, post-competition, we were able to execute a flawless course run, further validating the robustness of our design and implementation. Despite encountering challenges, our team's dedication to engineering excellence shone through, positioning us as a formidable contender in the realm of autonomous robotics.

Conclusion:

In conclusion we were able to break up this competition into multiple goals and deliver on them. We started with a MVP and built up to a robot that was fully capable of getting the maximum points available. We also represented the UNCA-NCSU Mechatronics program well by having the most capable robot proven by our first two rounds of scoring and our achievement of winning the Design competition. Most importantly we were able to collaborate as a true engineering team to accomplish the goals that we had set out at the beginning of the process.