Tokyo-bot: Emulating Industrial Automation

Erebus Oh , '24

Aaron Omadutt , '24

Dzineon Gyaltsen , '24

Tokyo-bot: Emulating Industrial Automation

Erebus Oh, Aaron Omadutt, Dzineon Gyaltsen

Advisor: *Matt Zucker*

Abstract

Our E90 project was to design, assemble, and program an omni-directional robot that integrated a web camera to intelligently sort (and ideally transfer from one location to another) colored boxes. The purpose of this project was to simulate an industrial workplace setting where similar types of sorting processes occur, and be able to optimize them. Developing a robot with high maneuverability combined with object detection capabilities could prove to be useful in optimizing workflows in warehouses and processing factories where items constantly need to be located and transported. With three group members, each with their own specializations, the project was divided up into mechanical prototyping/development, implementing swerve drive kinematics with software, and implementing object/color detection with computer vision software.

Introduction

For this project we essentially wanted to design a general-purpose robot that had a certain set of utilities. This accumulation of modular functionality would allow our robot to be applied in a variety of applications, such as auxiliary manufacturing processes, automated delivery systems, and intelligent product sorting. Our goal with this E90 project was to emulate manufacturing aiding robots in industry, but simplified into a “toy problem”. At its current state at the end of this E90 project, our robot is fully functional and is able to pick up cardboard blocks and sort them by color; this robot required the integration of swerve drive kinematics, computer vision methods, mechanical machining/assembly, and electrical wiring.

In this report we describe our motivations for pursuing this project, the design constraints we faced/design requirements we developed to address them, technical specifications for our project detailing the software and hardware we used, and a scheduling timeline for our project. We also include a brief description of our budget management process, technical challenges we faced along the way, and future applications of our work. We aim to provide two demos during our final presentation; the first demo will showcase the capabilities of the swerve drivetrain through a tele-operational demonstration, and the second will showcase the robot's autonomous features. The first demo will allow users to drive the robot, pick up boxes using the pickup mechanism, and deposit them in a designated spot via joystick controls. The second demo will highlight vision processing capabilities as the robot autonomously drives, detects colored boxes, and drops them off in a specified location with no user input.

Design and Constraints

Our mobile robot base has many constraints that must be considered in our design process. Physically, the robot is a 30.75'' square with four swerve drive wheel modules on each corner. The maximum speed of the robot will be limited due to the weight of all the robot components (chassis, wheels, battery, etc.). Thus, our application and demonstration of the robot should not require extremely high speeds that are unsustainable.

On the electronics side, the RoboRIO microcontroller supports up to 10 Pulse Width Modulation (PWM) ports, restricting the number of motors controllable via PWM. Additionally, the Power Distribution Panel (PDP) offers 8 ports each for 40A and 20A at 12 volts, limiting the overall power allocation to motor controllers. Additionally, the robot runs on a 12 volt battery which will limit our total uptime on the robot (estimated 5 minutes runtime for best performance).

Lastly, our $1200 budget for this E90 project imposed a significant constraint.

From these constraints and industrial inspiration for our project, we have come up with the following requirements to demonstrate an effective mobile robot with swerve drive and computer vision:

- Swerve Drive maneuverability
  - Translation: Drive forwards, backwards, sideways, and diagonally (1 meter each)
    - To assess accuracy, the robot's bumper will start at the origin of a precisely measured 1-meter tape line. Executing tele-operated movement, we'll measure the distance from the bumper's front to the tape endpoint. This data will determine the total traveled distance and quantify any undesired movement error.
  - Rotation: Perform a full 180° rotation both clockwise and counterclockwise
    - A straight tape line, aligned with the robot's center, will be placed on the floor. Another tape will mark the middle of the robot's front bumper, aligned with the floor tape. The robot will autonomously rotate clockwise 180°, and deviation from the robot's tape to the floor tape will be measured to assess rotational accuracy. The process will be repeated for counterclockwise rotation.
  - Autonomous: Drive autonomously in a rectangle
    - Drive towards a particularly colored box and stop within ½ ft distance from it. Overhead video footage will capture the autonomous movement, allowing for assessment of how accurately the robot centers and approaches the box.
- Computer Vision
  - Place outlines on colored boxes detected through the camera
  - Detect at least 3 objects in a frame at once

The codes we follow are according to the FIRST Robotics Challenge Robot Construction Rules via the 2023 Game Manual[1] (Page 74-111). In summary::

- robot weight must not exceed 125 lbs
- robot perimeter maximum 120 inches, and height maximum of 4 feet, 6 inches
- there must be an obvious and easy-to-access emergency stop button

Technical Discussion

Swerve drive, often referred to as independent steering, is an omni-directional drivetrain in which all wheels are independently steered and driven. This configuration allows the vehicle to spin on a spot, drive sideways, diagonally, or at other angles that aren't possible with traditional differential drivetrains.

Mechanical Prototyping / Development (Hardware)

The swerve modules we used (ThriftyBot ThriftySwerve[2]) were constructed and assembled prior to the start of the ENGR 90 project using a combination of 3D-printed and CNC-milled components. The main housing frame for each module was milled from aluminum (Figures 3 and 4), while components encasing the gears were 3D-printed using carbon fiber filament (Figure 2).

[1] https://firstfrc.blob.core.windows.net/frc2023/Manual/2023FRCGameManual.pdf

[2] Thrifty Swerve Assembly Instructions v1.3 Updated 05-03-2021.pdf

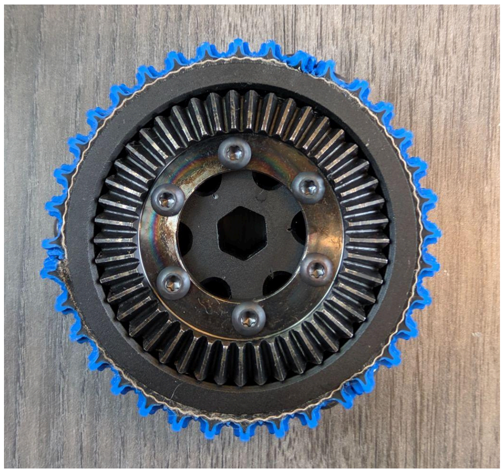

Figure 1. Side view of wheel encasing

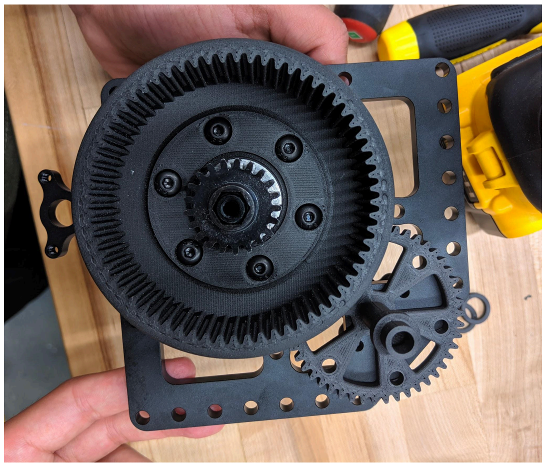

Figure 2. Inside view of gear

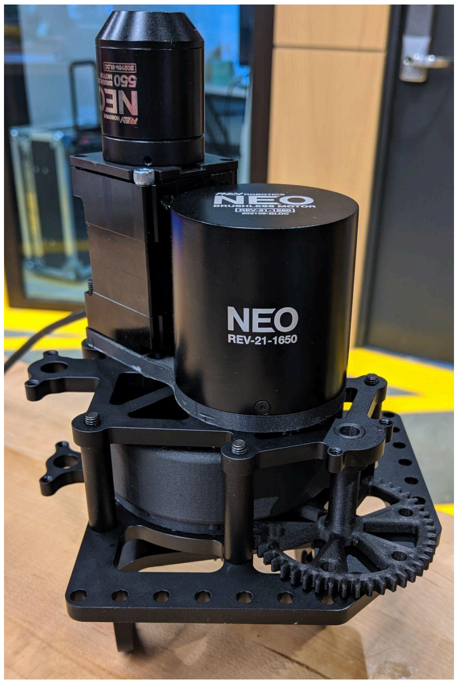

Figure 3. Motors attached to housing frame

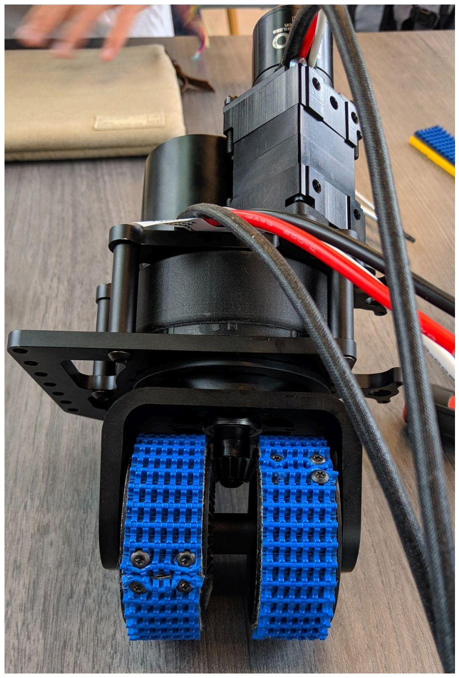

Figure 4. Fully assembled module

Each module is powered by two brushless motors: a NEO550 for wheel orientation and a REV NEO for wheel propulsion. The NEO550 is equipped with a 12:1 gearbox to enhance torque, ensuring uninterrupted wheel rotation without stalling while maintaining optimal rotational speed. Similarly, the REV NEO features a 12-to-21 teeth gear stage and a 15-to-45 teeth bevel gear stage, resulting in an overall drive wheel reduction of 5.25 to 1.

The estimated free speed of the robot, based on a 3” wheel diameter and motor gear ratios, was predicted to be 14.2 ft/s. However, the measured true speed of the robot is 10 ft/s, considering additional weight from the chassis and electronic components, as well as frictional forces.

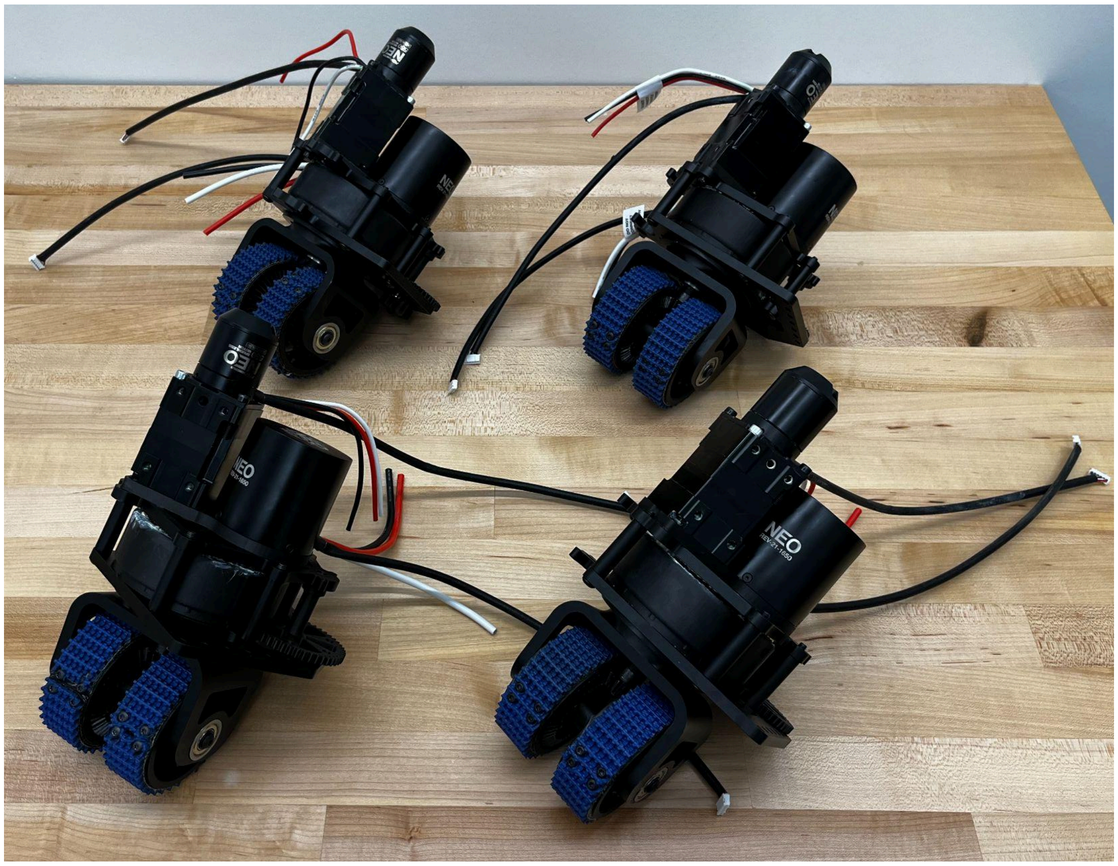

Figure 5. All four fully-assembled swerve modules

The chassis was fabricated out of four aluminum angle bars, each measuring 28.25 inches in length with dimensions of 3” x 3” x 0.375”. 45-degree cuts at the corners were made to reveal the swerve module wheels, facilitating operational visibility and streamlining troubleshooting protocols should a mechanical issue arise. To allow for linkage between bars, eight holes were drilled and tapped into the face of each corner, allowing the swerve module to be attached to the chassis itself. To enhance structural stability, two cross beams were secured across the center of the robot, minimizing potential bowing and vibrational effects while maximizing its load bearing capacity.

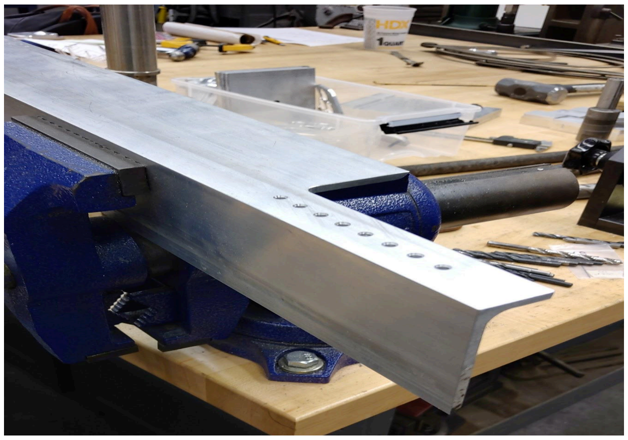

Figure 6. Close-up of one aluminum bar for chassis

In addition to corner modifications, the sides of the chassis were milled in a triangular pattern. This alteration contributed to both weight reduction and enhancing the robot's aesthetic appeal. The final assembled dimensions of the chassis are 30.75” x 30.75”.

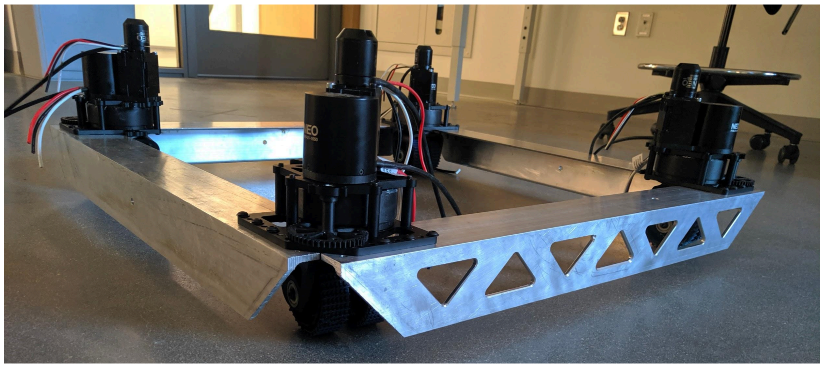

Figure 7. Swerve modules attached to chassis

A 0.125-inch plexiglass panel was mounted on and secured to the robot frame, serving as a housing enclosure for the electronic components.

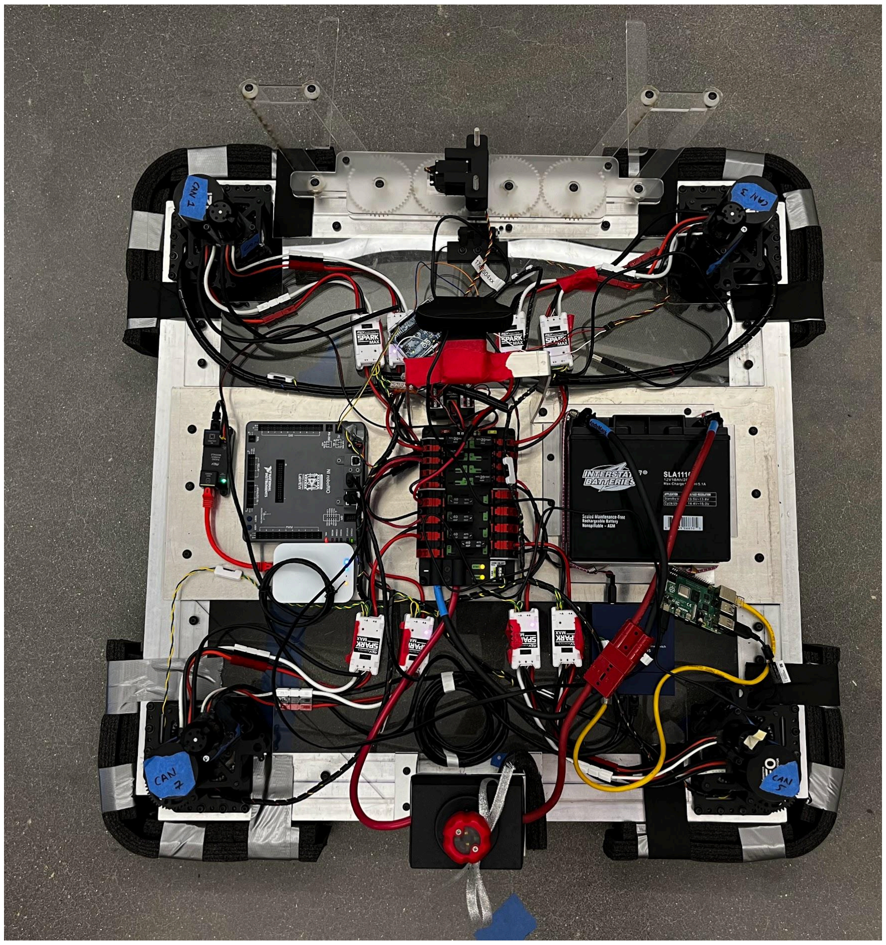

Figure 8. Wheel modules wired up to electronics

To pick up boxes of different colors and transport them to their designated drop-off locations, we developed a gripper mechanism. The design aimed to minimize the probability of mechanical failure, such as motor failures, by limiting the degrees of freedom. Additionally, to stay within our budget constraint, we opted to manufacture the mechanism from acrylic and utilized spare servos that were already available. The primary design features a four-bar mechanical system driven by gears controlled by a single servo motor for opening and closing. A secondary servo is used to pivot the entire mechanism up and down, which facilitates the lifting of the boxes off the floor once they are grabbed. All components were designed using Fusion 360 and laser cut in the makerspace.

The overall design phase was an iterative process. In V1 (seen in Figure 9), the proof of concept design was tested to verify the functionality of the four-bar mechanism for gripping the box. In V2, several improvements were made. The number of teeth on the gears was increased to reduce backlash, a protective cover was added for safety to prevent people's hands from entering the gears, and nylon spacers were used to hold the gears in place, reducing overall friction during gear rotation to prevent servo stalling. Additionally, a servo mount was added, and certain dimensions were adjusted to better fit the space available on the robot. The secondary servo, which controls the gripper's pivoting motion up and down, features 3D printed pin and slot parts. These components allow the mechanism to move smoothly up and down at the desired heights.

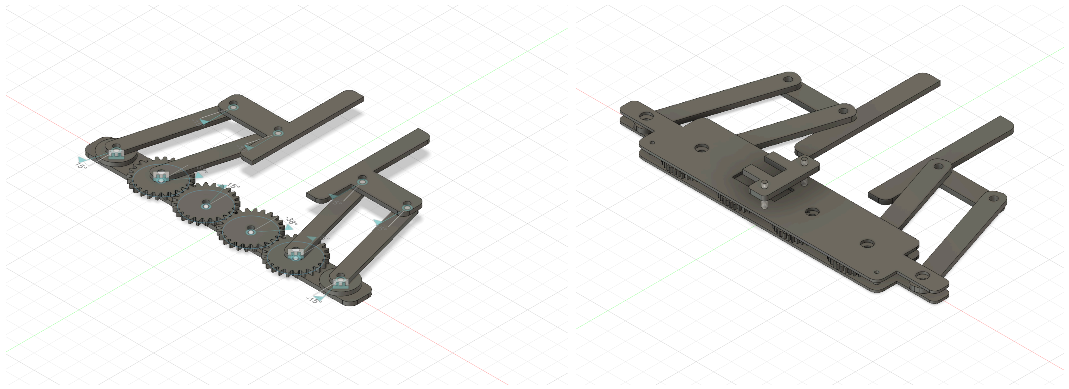

Figure 9: Fusion360 CAD Design of Gripper V1 (left) and V2 (right)

Electrical Prototyping / Development (Hardware)

Power for the entire system is provided through a 12 Volt, 18 amp-hour sealed lead acid battery which is connected to a breaker and a [Power Distribution Panel](https://www.vexrobotics.com/217-4244.html?q=&locale.name=English)[3] (PDP). The PDP we used was from Cross The Road Electronics' (CTRE), and its channels regulated power from the main battery to different subsystems including motors, motor controllers, and sensors. It is equipped with breaker slots for current

[3] https://www.vexrobotics.com/217-4244.html?q=&locale.name=English

protection and facilitates easy connection of wires to various electrical components through terminals or connectors.

Each module is equipped with an absolute encoder for rotational position tracking of the wheel as well as motor controllers for closed-loop velocity control to ensure each module's translation speed is in synchrony. The absolute encoder used was CTRE's CANcoder[4]. The CANcoder serves as a rotary magnetic encoder, establishing communication over the Controller Area Network (CAN) bus with the RoboRIO. This integration enabled us to continuously monitor the rotational position of the wheels accurately. All CANCoders were connected to a five-port WAGO splicing connector, allowing them to be powered through a single 12V/500mA port from the voltage regulator module[5] (VRM). The VRM was connected to the 10A breaker port on the PDP.

To achieve precise control over wheel speed, we used eight SparkMAX motor controllers[6]. This choice was due to their compatibility with the REV motors in our current setup. The controllers come equipped with built-in closed-loop velocity controllers, enabling us to efficiently regulate the speed of each module. All motor controllers were connected in series over the CAN bus. They were wired to link to a designated slot on the PDP, with a 40 amp breaker for controllers managing the NEO motors and a 20 amp breaker for controllers handling the NEO550 motors.

All encoders were connected through the CAN bus, which is run directly to the RoboRIO. The RoboRIO[7] (Robot Input/Output) is a compact controller that features various input and output ports, including digital and analog I/O, PWM (Pulse Width Modulation) outputs, USB ports, and more. The roboRIO serves as the central processing unit for the robot, running the robot code and handling communication with other devices and sensors.

---

[4] https://store.ctr-electronics.com/cancoder/

[5] https://www.vexrobotics.com/217-4245.html

[6] https://www.revrobotics.com/rev-11-2158/

[7] https://www.ni.com/en-us/shop/model/roborio.html

On the gripper and network architecture electronics framework, a Open-Mesh OM5P-AC radio[8] was employed to create a local area wifi signal that allowed us to wirelessly connect to the robot for both deployment of code as well as tele-operational and autonomous control. The radio was connected via a Power Over Ethernet (POE) connection to supply power to the radio from the VRM and data transfer to the roborio. Additionally, an Arduino UNO Rev3[9]was wired via SDA and SLA to the Roborio to set up an I2C connection for controlling the HS-5685MH servo motors[10]. The servos were controlled via PWM connection to the Arduino, but were powered from the 2A/5V ports on the VRM.

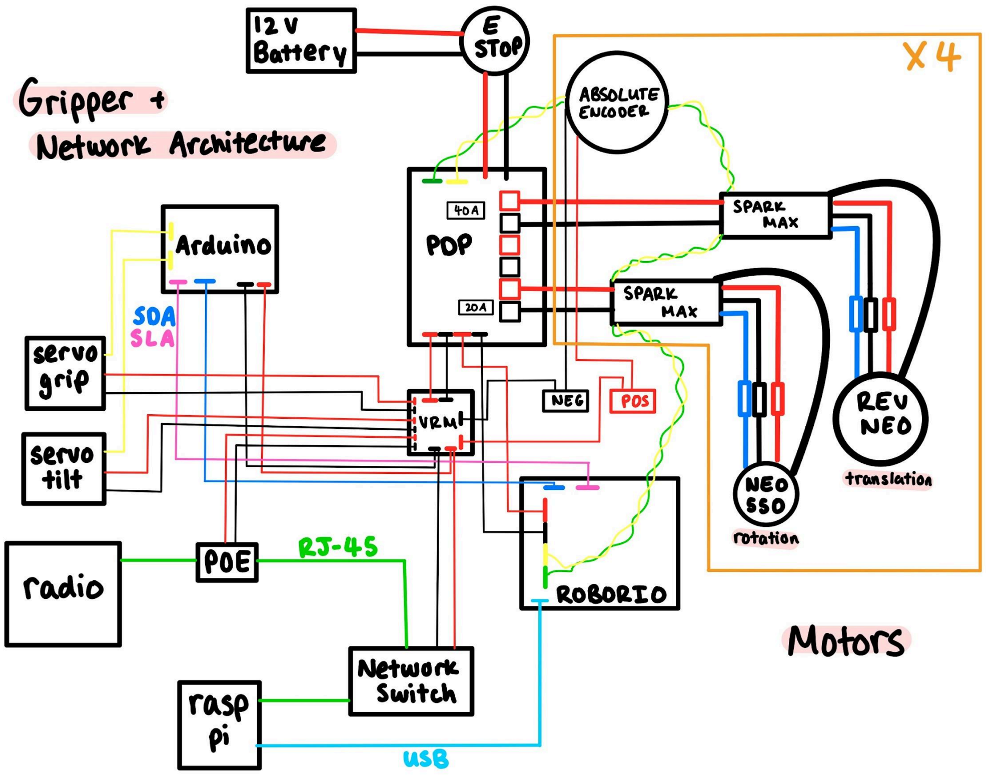

Figure 10. Electronic wiring schematic for one wheel module

8 https://www.andymark.com/products/open-mesh-om5p-ac-dual-band-1-17-gbps-access-point-radio

9 https://store.arduino.cc/products/arduino-uno-rev3

10 https://hitecrcd.com/products/servos/discontinued-servos-servo-accessories/discontinued-digital-servos/hs-5685mh-high-torque-hv-metal-gear-servo-/product

On top of the mechanical and electrical components necessary for this project, we had to utilize and integrate several software packages and components. Software for our project was split between components that were necessary for implementing swerve kinematics and those that were necessary for object/color detection.

Swerve Drive Implementation (Software)

Components

- PS4 DualShock 4 Controller[11]
    - The controller that will be used for teloperating our robot has two analog sticks, a D-Pad, 4 Game buttons, and a six-axis motion sensor. More specifically, the left joystick will control translation of the robot while the right joystick will control rotation. Since the controller is similar to most console game controllers, it will be easy for users to learn.
    - Amazon Link[12]
- WPILib / FRC Game Tools
    - The software environment and packages for our project will largely be through FRC. FRC's WPILib[13] is the standard software library for code to be written and deployed to our robot, and FRC's Game Tools[14] is a software bundle (provided by National Instruments[15]) that provides a control interface for managing and configuring our robot. The FRC Game Tools bundle includes critical software

[11] https://direct.playstation.com/en-us/buy-accessories/dualshock4-wireless-controller
[12] https://a.co/d/fFxc5Qi
[13] https://github.com/wpilibsuite/allwpilib/releases/tag/v2023.4.3
[14] https://docs.wpilib.org/en/stable/docs/zero-to-robot/step-2/frc-game-tools.html
[15] https://www.ni.com/en/support/downloads/drivers/download.frc-game-tools.html

components including the FRC Driver Station, FRC roboRIO Imaging Tool and Images, and FRC Utilities.

- Phoenix Tuner X[16]
    - The Phoenix Tuner X software configures the CAN Coders of each wheel module. This software is provided by CTRE.
- REV Hardware Client[17]
    - The REV Hardware client is designed to configure and test the SparkMAX motor controllers, two per wheel. Specifically, to ID the motor controllers to be used in code.

For our swerve drive implementation, we provide a thorough review of swerve drive kinematics. The math behind swerve drive kinematics is useful for both tele-operation mode (free-range joystick driving) and autonomous mode (autonomous driving).

Tele-Operation Mode

For tele-operation, we mapped user input through the joystick to a set of wheel velocities and angles for each wheel module. The first step of this process involved defining a coordinate system and orientation for the robot frame[18] that made sense for the user and remained consistent throughout. The joystick inputs naturally lie within a range of [-1, 1], but to get smoother motion we filtered out small values between -0.1 and 0.1 by setting them to 0. The joystick inputs were mapped such that the left joystick corresponded to translation and the right to rotation; pushing the left joystick towards the right corresponds to positive *x*, and pushing it down corresponds to positive *y*. Pushing the right joystick to the

[16] https://pro.docs.ctr-electronics.com/en/stable/docs/tuner/index.html
[17] https://docs.revrobotics.com/rev-hardware-client/
[18] Coordinate system that the robot perceives and operates under

right corresponds to positive *z* (or θ for angle), with no impact when pushed up or down. The joystick inputs are shown in Figure 11, and the mapped chassis outputs are shown in Figure 12.

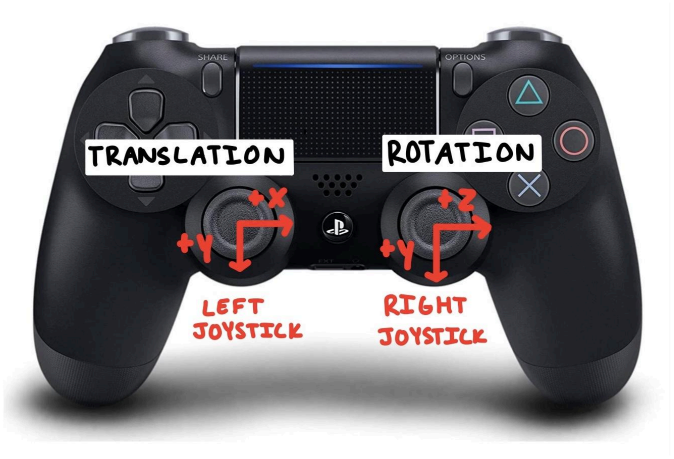

Figure 11. Joystick inputs mapped to robot translation and rotation

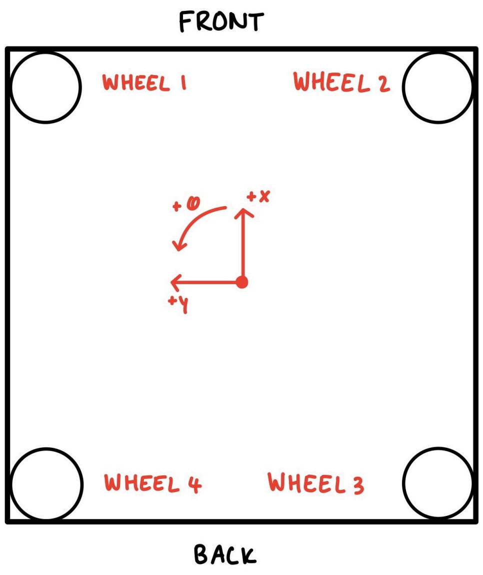

Figure 12. Inverted orientation of joystick inputs overlaid on top of chassis

To the user, it would make more sense that pushing the left joystick up would correspond to the robot moving forward and pushing to the left would correspond to the robot moving leftward. Similarly, a leftward push of the right joystick should correspond to the robot rotating to its left. We implemented these changes in our code by simply inverting the inputs.

Now, we run into the problem of converting these robot body (chassis) speed values to individual wheel linear velocities (V_{i_wheel}) and wheel orientations ($\boldsymbol{\alpha}_{i\_wheel}$) for each of the four swerve wheel modules. Linear and angular velocities are the translational (meters/second) and rotational (radians/second) speeds at which the robot is traveling, respectively. Each wheel module has both a drive motor and an orientation motor, which we must set to a certain speed or position value in code. To find these values, we thought of the desired chassis linear and angular velocities as input vectors. Using this approach, it becomes easy to calculate the vector sum of these inputs and conceptualize the resultant vector's (green) direction as the orientation of the wheel module and magnitude as its drive speed. A visualization of these vectors are shown in Figure 13.

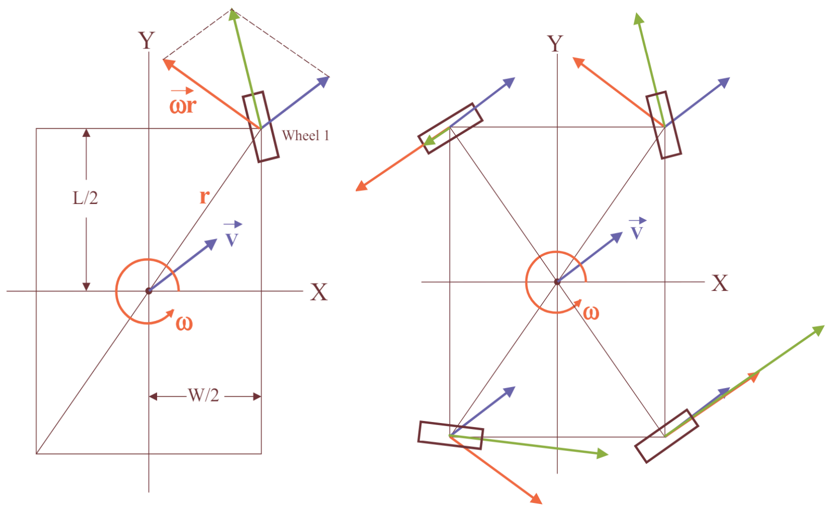

Figure 13. Vector components of each wheel, Ether, Chief Delphi

This approach is effective since removing either the translation or rotation input results in simple, predictable movement. If no robot translation is specified by the user via joystick the robot will simply rotate in place about its center, and if no robot rotation is specified the robot will simply glide/translate. When combining these inputs, it becomes important to note the magnitudes of each input vector. Since the tangential velocity of a rotating body is defined as the product of its distance to the center point about which it is rotating and the angular velocity at which it is rotating, the magnitude of the angular velocity component vector (orange) for a given wheel is defined as $\omega * r$ where ω is the desired chassis angular velocity and r is the distance from the center of the respective wheel to the center of the robot. The magnitude of the linear velocity component vector (blue) for a given wheel is simply the desired chassis linear velocity.

Now, we can directly set the translation motors for each wheel to the magnitude of its respective resultant vector (a speed value in m/s), and set the rotation motors for each wheel to the angle formed by the resultant vector (a position value in degrees) in accordance with the coordinate orientation specified in Figure 12. With this last step we now have four sets of wheel linear velocities and wheel angular positions, and swerve drive kinematics is fully implemented.

<u>Autonomous Mode</u>

At a high level, a similar problem must be solved for autonomous driving as in the tele-operation mode with regards to setting translation and rotation motors to distinct wheel linear velocities and angular positions. In autonomous mode, the motion of the wheels is determined by the information received through the webcam instead of through a joystick. For our autonomous demo, we need the camera to detect distinct colors and outline distinct rectangles to know what objects are in the robot's field of view. Through a process known as visual servoing, we used this visual data to rotate the robot such that the object of interest is centered from the robot's point of view and translate the robot such that the object of interest is close enough to be picked up by the gripper mechanism. The control law that governs this object centering motion is given in Equation 1, while snapshot frames from the camera before and after visual servoing are shown in Figure 14.

Diving deeper, we can break up and explain the left/right rotation motion and the forward/backward translation motion separately.

$$\dot{\theta} = k_p \cdot \Delta x \qquad (Eq.\ 1)$$

For the rotational aspect where the robot is centering the object of interest in its field of view, we can explain each component of equation 1 and how it relates to the diagram shown in Figure 14. Equation 1 essentially states that the speed at which the robot should rotate is equal to some constant of proportionality k_p multiplied by the pixel difference in distance along the *x*-axis from the center of the object to the center of the frame. For our webcam, we experimentally determined a small k_p value that worked fairly well for our purposes. We also implemented a small deadband of 20 pixels so as to prevent small oscillations while maintaining an accurate centering. Now, to actually set our wheel linear velocities and angular positions to achieve this we simply set each wheel's rotation motor to 90 degree offsets of each other, and each wheel's translation motor to the calculated product of k_p and Δx.

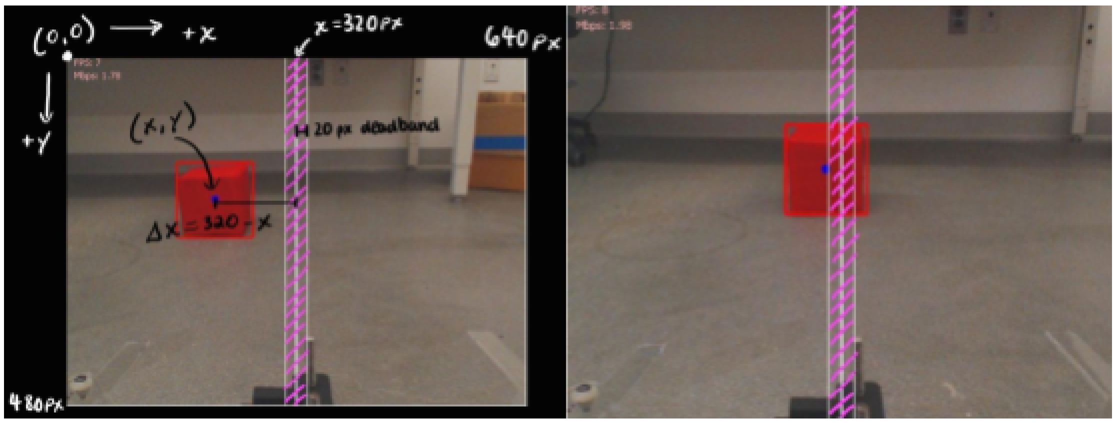

Figure 14. Visual servoing before and after images

For the translational aspect, we can observe how the area that the object of interest would take up within the camera frame would increase as the robot approached the object. Given that the object is centered, we can thus set a minimum threshold for the area such that the robot is close enough to the object to pick up. This type of motion is purely translational, as the robot should slowly approach the object at a constant speed until the threshold is reached. As such, we can set each wheel's rotation motor to 0 degrees (directly straight) and each wheel's translation motor to some arbitrarily slow speed.

For autonomous kinematics, visual servoing in combination with state machine controls is all that was implemented. A state machine diagram for our autonomous demo is shown in Figure 15.

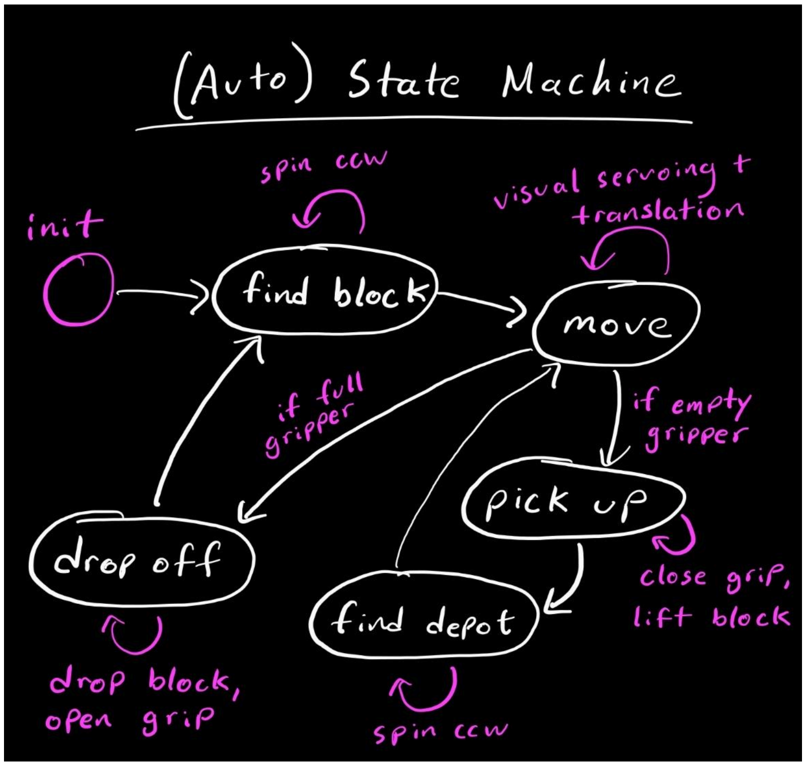

Figure 15. State machine visualization for autonomous demo

Object / Color Detection Implementation (Software)

Components

- Logitech C920 HD Pro Webcam[19]
    - This web camera records in HD 1080p quality at 30 frames per second, and connects via USB A to the RoboRIO. It also has a 78° field of view, automatic light correction, records at 3 megapixels, and is relatively lightweight at 162 grams. These specifications will support our computer vision tasks such as color detection.

[19]https://www.logitech.com/en-ch/products/webcams/c920-pro-hd-webcam.960-001

- Raspberry Pi[20]
    - The Raspberry Pi 4B is the coprocessor for computer vision, which offloads computational resources on the roborIO. With 64 GB of storage and a custom FRC raspberry pi image, our python scripts for vision are loaded onto the pi, which then communicates the information via network to the roborIO.
- Netgear Fast Ethernet Switch[21]
    - Enables network communication between the roborIO, raspberry pi, and the computer.

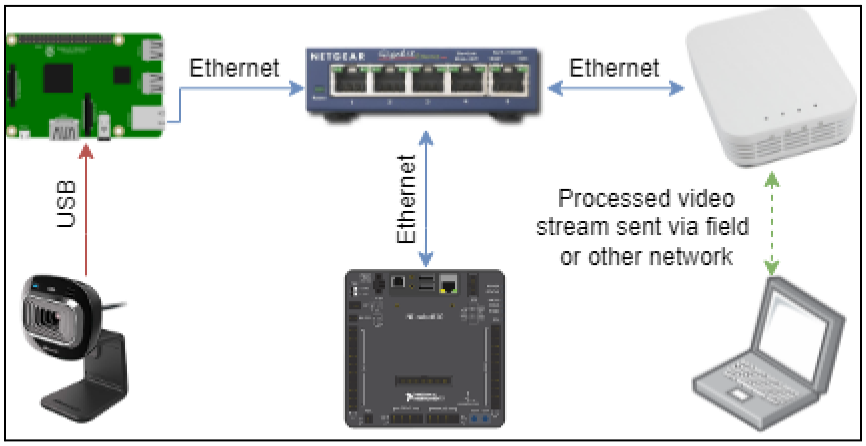

Figure 15. Raspberry Pi as vision coprocessor network system. (FRC Docs)

For our computer vision component of the project, we integrated a color and object recognition system by programming a high-resolution webcam. Both the vision system and the drivetrain work together to enable our robot to autonomously detect and sort colored boxes in an enclosed environment, mirroring industrial automation scenarios.

We programmed our webcam using a Raspberry Pi, which is connected via USB to our webcam, and then also connected to the robot via ethernet (through ethernet cable and ethernet switch), as seen

[20] https://www.raspberrypi.com/products/raspberry-pi-4-model-b/
[21] https://www.bestbuy.com/site/netgear-gigabit-ethernet-switch-blue/=aw.ds

in Figure 15. Having the computer vision code run primarily on the coprocessor allows for more intensive vision processing that would have been limited on the roborIO.

Given the images from the camera stream, we use a five step process in order to accurately detect colored boxes' locations and areas, as seen in Figure 16.

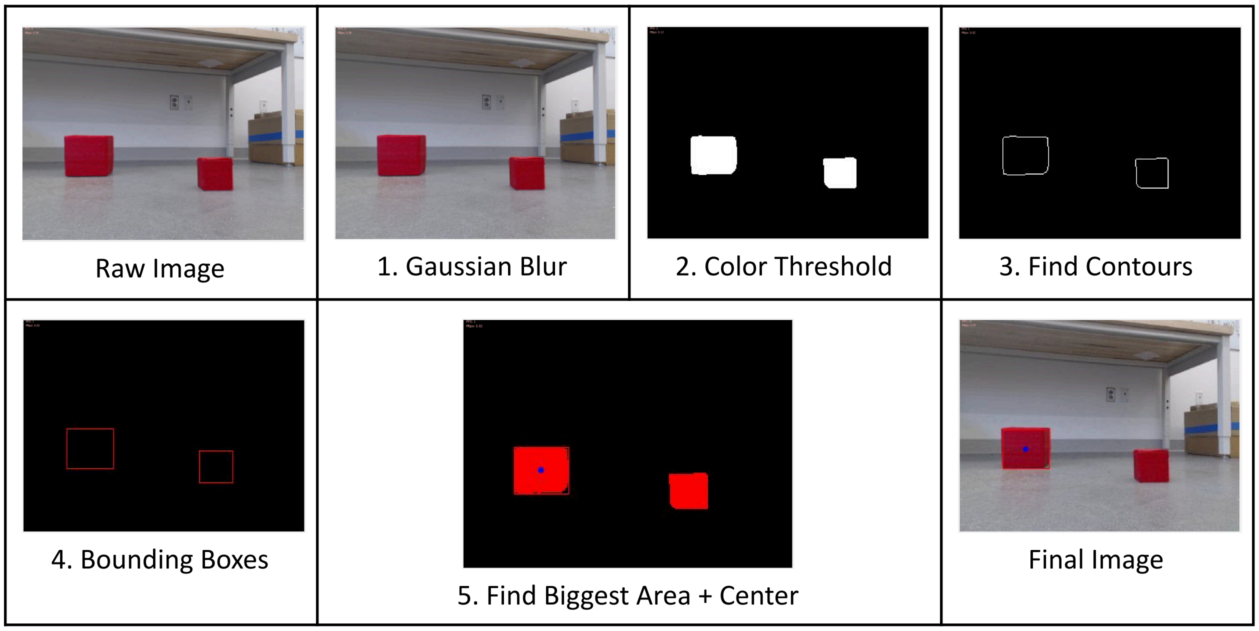

Figure 16. Computer Vision color detection process.

1. Apply <u>Gaussian Blur</u>.
    a. 7x7 kernel
2. <u>Color Threshold</u> for desired colors (red, yellow, green).
    a. Hue-Saturation-Value (HSV) colorspce
3. <u>Find contours</u> to detect objects.
4. Create <u>bounding boxes</u> around detected contours.
5. Calculate the biggest <u>object area</u> and its <u>center</u> location (x & y pixels).

The first step, the Gaussian Blur, efficiently removes random noise which greatly helps object detection accuracy–especially in environments with inconsistent lighting and budget cameras. We found a 7x7 kernel was optimal for both speed and noise removal. Other noise removal techniques, such as morphological operators, are also effective but are computationally expensive and slower.

The next step, Color Thresholding, filters every pixel in the image. All pixels that are contained within the defined color range (for example, all red pixels) become white, and otherwise black. The resulting image of color thresholding is a black-and-white binary image, which is also called a mask. This thresholding can be done in numerous colorspaces, or ways to define color in an image. The most common one, the RGB colorspace, means every pixel is defined by the number of red, green, and blue present. For our detection algorithm, we found Hue-Saturation-Value to be a more effective colorspace, which is visualized in Figure 17.

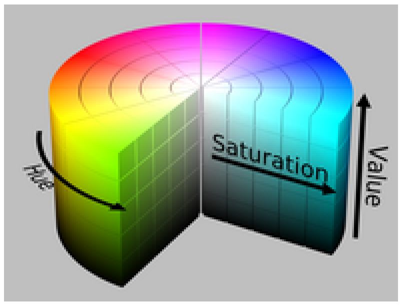

Figure 17. HSV Colorspace ([Wikipedia](https://en.wikipedia.org/wiki/HSL_and_HSV)[22])

The third step of our vision process is finding contours. A contour is defined as a continuous bounded shape outline separating colors. Since the color thresholded image is black and white, this allows OpenCV's findContours() method to work effectively. This method is well-used in finding borders of objects.

[22] https://en.wikipedia.org/wiki/HSL_and_HSV

The fourth step is creating bounding boxes around the contours we found in the previous step. Since most of our target objects are blocks, a bounding box is a good approximation. Additionally, calculations for area and location are significantly easier with boxes than irregularly shaped contours.

The final step is calculating bounding boxes' areas and center locations. For purposes of our application, we are only interested in the biggest target block. With the bounding boxes given from step four, we can easily calculate the area ($Area = Height \times Width$) and the center ($x = Top\ Left\ Corner\ X + \frac{1}{2}Width,\ y = Top\ Left\ Corner\ Y + \frac{1}{2}Height$). We have successfully detected the location and area of an object.

All the vision processing is performed on the raspberry pi, which then communicates the calculations to the robot by publishing these values to the network. The roborIO in turn subscribes to these values being updated, and can be used in code for visual servoing (as described earlier).

Results / Design Evaluation

In summary, we were able to achieve the design requirements we set out for ourselves at the start of our E90 project. We were able to successfully program a dynamic robot with swerve drive maneuverability, object/color detection capabilities, and autonomous driving. Specifically, we were able to translate the robot by driving it forwards, backwards, sideways, and diagonally 1 meter each with only limited human error while driving. This demo can be seen in this video. We were also able to rotate the robot in place 180 degrees both clockwise and counterclockwise, as seen in this demo. For our autonomous driving design constraint we were able to autonomously navigate to exactly half a foot from a colored box, as evidenced by this demo. A successful run-through of our autonomous finite state machine is shown in this demo, showcasing our ability to pick up and sort objects. Lastly, Figure 18 shows the results of vision processing performed by our robot. The figure clearly shows that our robot is able to distinguish between the different colored objects within the camera's field of view as well as appropriately delineate between objects of varying sizes.

Thus, we have met all of our desired design requirements and are content with our final iteration of the robot. Some improvements can be made to make the robot's autonomous navigation and gripping mechanism more robust, but for the purposes of our E90 we believe that we have successfully demonstrated our abilities to design around and think critically through project constraints.

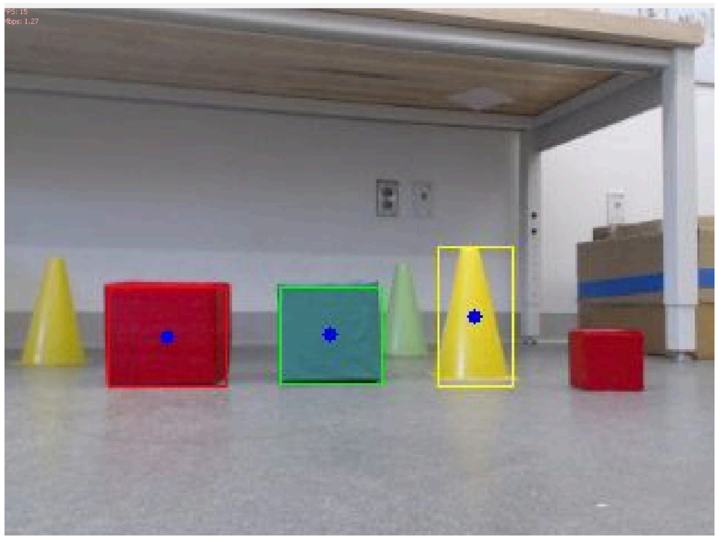

Figure 18. Snapshot showcasing object/color detection vision processing

Project Plan

<u>Timeline</u>

Project Milestone, E90 Due Date

Week	Date	Description
0	12/23/23 Fall 2023 Semester	Finalize design requirements Finish robot hardware base Integrate controller input Start testing camera

		Implement swerve drive code
1	1/21/24 - 1/27/24	**Prototype - Basic Swerve Drive & Vision** ● 4 Wheels wired up ● Controller, Vision ● Teleop Swerve Drive
2	1/28/24 - 2/3/24	Start Swerve drive for autonomous Finalize & mount camera location Program vision nearest block color detection
3	2/4/24 - 2/10/24	Testing, Debugging
4	2/11/24 - 2/17/24	Testing, Debugging
5	2/18/24 - 2/24/24	Testing, Debugging
6	2/25/24 - 3/2/24	Run autonomous on simple path Program object classification vision component
7	3/3/24 - 3/9/24	Implement more complex autonomous path, vision results dependent
8	3/10/24 - 3/16/24	***Spring Break***
9	3/17/24 - 3/23/24	Autonomous code flow finalized, testing
10	3/24/24 - 3/30/24	**3/29: Mid-Semester Project Presentation Recording** Record practice presentation Testing, Debugging
11	3/31/24 - 4/6/24	Testing, Debugging
12	4/7/24 - 4/13/24	Testing, Debugging Documentation
13	4/14/24 - 4/20/24	**Core Functionality - Delivery Sorting System** ● Mount electronics ● Autonomous delivery system ○ World Frame Mapping ○ Swerve Drive in Autonomous ○ Vision block sorting

		● Advanced vision features ○ object detection, color, location Testing, Debugging Documentation, write report
14	4/21/24 - 4/27/24	**4/26: Draft Report** **Project Extensions** ● Intake/Gripper ● LED and/or LCD Testing, Debugging
15	4/28/24 - 5/4/24	*5/3: Classes End* Record robot performance Practice presentation Finalize code, documentation
16	5/5/24 - 5/11/24	**5/6, 5/7: Presentations** **5/10: Final Report**

Project Cost

With three group members, we have a total budget allocation of **$1,200.** We have a working spreadsheet[23] where we keep track of our material/ equipment purchases, which we will periodically validate with Cassy Burnett. For our project costs it should be noted that we started the project with a rough base already constructed from the robotics club, which is not accounted for in the spreadsheet. Prior to E90, we consulted with J. at the machine shop to assist in some metalwork related to constructing the chassis of the robot.

[23]https://docs.google.com/spreadsheets/d/17t86gaUFV7Hk1g97KqLNI7NI1hDnBZ1pXmSWMVdARrY/edit#gid=0