P.A.R.R.O.T: Parallel Asynchronous Robots, Robustly Organizing Trucks

Authors: Prithu Pareek, Omkar Savkur, Saral Tayal
Affiliation: Electrical and Computer Engineering, Carnegie Mellon University

Abstract—**This project aimed to design and build a multi-agent robotic system capable of transporting pallets from a warehouse floor onto delivery trucks in an efficient manner. Due to the limitations of time and resources of this course, we built a scaled-down version of this system on a 1- x 2-meter field with 10 x 12 cm robots. While the physical size is not representative of real-world systems, we hoped to design algorithms for computer vision, planning, and communication that result in increased task-completion efficiency in a multi-robot system that can be directly used in real-world systems. We tested our system on these metrics and found in 3x speedup in task-completion time with 3 robots when compared to 1 robot for the same task.**

Index Terms—**Computer Vision, Controls, Design, Efficiency, Multi-Agent, Planning, Robot, Warehouse**

1 INTRODUCTION

Warehouse productivity is inversely proportional to how long it takes to load/unload a truck. As demands on warehouses grow, the first step to increasing warehouse productivity is often automating warehouses with robots. However, while humans might be able to move around each other fairly intuitively, having multiple robots running around a warehouse in an efficient and collision-free manner is quite difficult [4]. Companies such as Tesla Factory robots [15], Amazon Robotics [1], Geek+ [11], and Caja Robotics [12] are just a few of the companies deploying swarm robots to factories and warehouses. This capstone project explored some of those challenges from both a technical and systems perspective.

2 USE-CASE REQUIREMENTS

The use-case requirements for our "sandbox" version of the multi-robot warehouse organizing system can be divided into three overarching sections: pallet pick-up/drop-off, run-time and latency requirements, and multi-robot scaling efficiency.

2.1 Pallet Pick-Up / Drop-Off

In order to complete their core task of warehouse organization and truck-loading, our robots must be able to pick up, drop off, and transport pallets around the warehouse floor. In order to do this, there are a few things that are necessary. It is extremely important that our robot be able to reliably (**<1% failure rate**) latch onto and let go of the pallets when the robot is at the pallet position. Our system must also be able to drop off the pallets in the correct locations with an **accuracy of 5mm**. Finally, we want the robots to be able to load the trucks in a specific order optimal for the trucks to unload as they make multiple stops on their route.

2.2 Run-time and Latency

Our robots must have a **battery life of 4 hours**. This was calculated based on the average length of a workday being 8 hours with a break for lunch in between. We don't expect warehouses to operate completely without people, and thus want the robots to be able to last for half a shift/charge during the lunch break. We would also like a **latency of under 5 seconds** between the initial start command being sent to the robots and the first movement.

2.3 Multi-Robot Scaling Efficiency

The primary selling point of our system is the efficiency provided when using a multi-robot system rather than a single robot. The following sub-requirements address this efficiency requirement. Arguably the most important is that our robots must not collide with each other, or any other obstacles in the environment. This is for safety reasons. Finally, we require a **speedup of 2x when using 3 robots** when compared to 1. This number was estimated using Amdahl's law, presented in (1), which is an equation used to determine the maximum speedup possible in parallel computation applications.

$$\text{speedup}(f, n) = \frac{1}{(1-f) + \frac{f}{n}} \tag{1}$$

In this case, f is the fraction of the work that can be parallelized and n is the number of robots. We assume that 25% of the work is inherently sequential due to contention between the robot's paths.

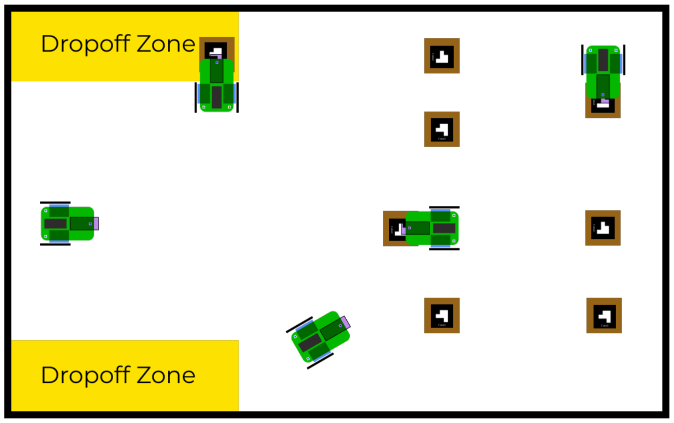


Figure 1: This is the top view of the field. The pallets are depicted in brown, and the robots are depicted in green.

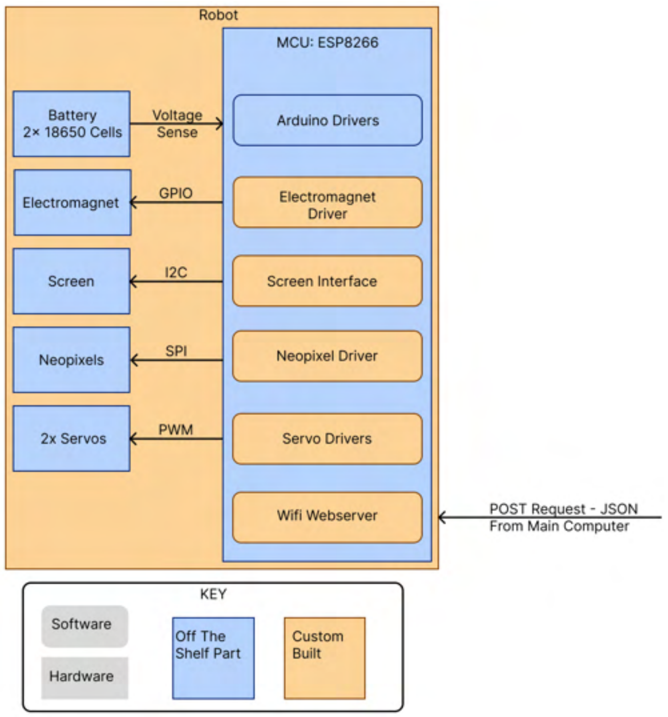


Figure 2: Robot block diagram.

3 ARCHITECTURE AND/OR PRINCIPLE OF OPERATION

Our system includes a field that the robots and pallets are moving around on, an overhead camera viewing the entire field, and a main centralized computer that is doing the localization, path planning, and controlling for each of the robots. Fig. 1 depicts a cartoon version of what the camera will see.

Each robot is constructed out of a PCB. Each one has an ESP8266, which waits for commands via HTTP POST from the main computer and interfaces with the peripherals appropriately. Fig. 10 depicts how the robot should behave at a block diagram level. The robots are meant to be as simple as possible, with all of the computation being done by the main computer. The robots have an Aruco fiducial marker for localization, an electromagnet for picking up the pallets, batteries for power, and two continuous rotation servos for movement. The robots initially had Neopixels for localization and a screen for debugging, but we swapped them to fiducials for higher accuracy and reliability, as further described in Section 6.2.3.

The main computer, whose subsystems are shown in Fig. 3, takes the camera feed in, uses OpenCV to localize the robots and the pallets, and has a task planner to allocate tasks to robots. From there, each robot has its own planner that plans a path that does not collide with any obstacles or robots (moving or stationary) in the field. Each robot also has its own controller that keeps the robot along its specified path. There is also an emergency stop if the user wants to stop the robots. Originally, the computer vision subsystem would trigger the emergency stop if it determines if there will be a collision, but we did not have enough time to implement that feature and opted for a manual emergency stop.

Warehouse managers will interact with this system through the main computer side which will allow them to run the system, which will take into account an arbitrary number of pallets and robots. From there, the main computer will determine how many robots are present on the field, where the robots and pallets are, assign each robot to move a specific pallet, plan paths for all the robots, and then send servo and electromagnet commands to the robots to get them to follow their paths. The main computer also has a visualizer that shows the paths of the robots and the expected positions.

4 DESIGN REQUIREMENTS

The Design Requirements for our “sandbox” version of the multi-robot warehouse organizing system are built off of our Use-Case requirements and can be divided into the same three overarching sections: pallet pick-up/drop-off, run-time and latency requirements, and multi-robot scaling efficiency.

4.1 Pallet Pick-Up / Drop-Off

Since the user needs the system to accurately drop off pallets in the correct location with 5mm precision, our system needs to be able to detect all the robots and pallets on the field with 100% accuracy and be able to differentiate between them. The system also needs to be able to localize the robots to within 5mm so that we can guarantee the pallets carried by the robots will be delivered within 5mm of the desired location. A standard HD USB webcam with a resolution of 1080 by 1920 must have a **maximum error of 5 pixels** difference between where the localization system thinks the robots and pallets are compared to their

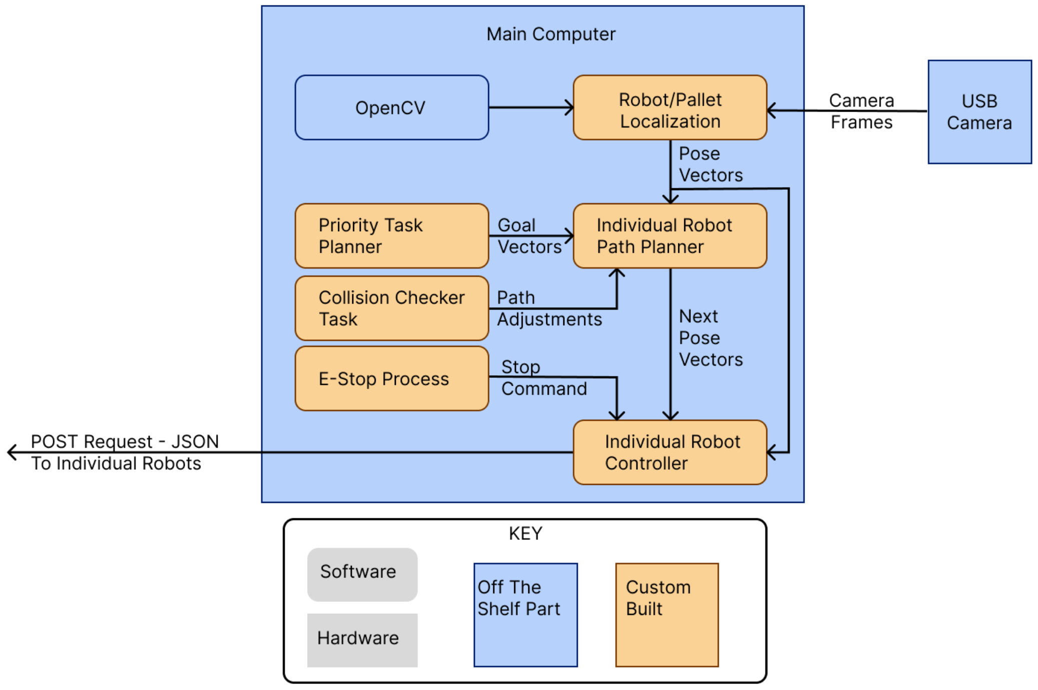


Figure 3: Main computer's software stack block diagram.

actual positions in order to achieve a 5mm localization accuracy for a 1m by 2m "sandbox".

The electromagnet must have enough holding force to pick up the pallets. A 3cm by 3cm pallet made out of 3D printed material with a steel top will have a mass of 29.49g as shown in (2). Therefore, the electromagnet must have a holding capability of 0.290N.

$$\begin{aligned} m = V\rho = (3\text{cm} \cdot 3\text{cm} \cdot 2\text{cm})1.25\text{g/cm}^3 + \\ (3\text{cm} \cdot 3\text{cm} \cdot 2\text{cm})7.7\text{g/cm}^3 = 29.49\text{g} \end{aligned} \tag{2}$$

The robot servos must also be sized such that they have enough torque to propel the robot and the pallet forward. We estimate that the robot will have a mass of 172.72g (48g for 18650 30Q cells, 1.72g for the MCU, 11g for servos and wheels, 26g for the electromagnet, 36g for the PCB, and 50g maximum for any miscellaneous components). Therefore, the servos must propel a minimum mass of 172.72g and a maximum mass of 202.21g.

In order to load pallets onto the truck in the correct order, the task planner must assign robots to pallets in such a manner that the pallets must arrive at their desired position at the correct time. The path planner must adhere to the task planner's ordering requirements.

4.2 Run-Time and Latency

In order to achieve 4 hours of battery life on the robots, the battery capacity must supply 4 hours' worth of power drawn by the electromagnet and the servos, which are the two components that draw the most current. Equation (3) describes how the battery life t, total current draw A, and battery capacity C are related, where N is the number of components drawing current and A_i is the current draw of component i.

$$t = \frac{C}{A} = \frac{C}{\sum_{i=1}^{N} A_i} \tag{3}$$

In order for the user to perceive the robots as having a smooth motion, the communication latency between the computer and the robots must be under 100 ms, as this is on the order of magnitude for WiFi communications. Furthermore, we require a maximum of 200 ms time for the sense-plan-act loop. This is so that once the robots are moving, they can do so at a fast speed, without compromising correctness. The human reaction time is just under 300 ms, so the sense-plan-act loop and communication latency should take under 300 ms so that the robots seem to move smoothly on their path.

4.3 Multi-Robot Scaling Efficiency

One of our paramount requirements is that the robots have 0 collisions with robots and other obstacles in their environment. We require a motion planner planning in space and time that generates paths such that robot paths do not intersect at the same time. To aid with this 0 collision metric, each robot should also be within 5mm at all times from its expected point in its trajectory. We will also need an emergency-stop fail-safe to prevent any imminent and potentially unavoidable collisions.

5 DESIGN TRADE STUDIES

5.1 Robot design - PCB

Swarm robots like ours (Tesla factory robots, Amazon Robots, Crazyflie drones, etc) are designed to be as simplistic and processing-light. This is to achieve scaleable production where it is imperative to bring the cost of each robot down [5]. Similarly, we too approached building our robots as simply as possible. Our robot's frame is our PCB and as such it doubles as an electrical interconnect between our electronics and as a structural member. This greatly simplifies our design when compared to a more mechanical robot with real cables/wires running between components. This PCB approach also makes our robots consistently similar which is especially important for localization (discussed later in the paper)

The PCB is also designed to have minimal active electrical components on it and rather focuses on acting as an electrical bridge between off-the-shelf modules that attach to the PCB. Such modules include our MCU, our screen, servos, batteries, power regulator, etc. This makes the assembly of the robot significantly easier while also making our robot easy to manufacture (less reflow work), and also reduces the scope of the electrical design of the robot – which is not the focus of this project.

5.2 Robot design - why Computer Vision?

Localization on robots is typically done via a combination of different sensor systems and fusing them in a technique called "sensor-fusion".

Since the goal of our project is centered around the scaleability of multi-agent robotic systems and we have budget/time restrictions, we decided to use just one sensor system – computer vision – for our localization as opposed to a wheel encoder system.

This brings a few advantages to integration. Firstly, computer vision enables us to not only localize our robots, but also localize our pallets via simple QR-fiducials, greatly simplifying pallet detection. Secondly, it reduces the BOM cost of each robot since we don't need to include an extra encoder. Thirdly, it reduces timing issues and network latency delays from an extra loop of getting feedback from the robot and accounting for the time-delay staleness of the encoder data in the control loop.

However, computer vision does come with pitfalls, especially around making sure that the environment is appropriately lit and making sure no obstructions interfere with the camera's vision of the robot.

While encoders, especially when fused with computer vision, could make localization more precise and help us meet our use-case requirement, our preliminary testing shows that vision-only localization is still providing us with more than enough accuracy at around 2-3mm.

5.3 Robot design - electromagnet vs mechanical pickup

Most robots that operate in warehouses use a mechanical forklift-style system to physically pick up pallets to move [9]. Given our scaled-down robot sandbox, goals for better understanding multi-agent algorithms, and time/-cost limitations, we decided to simplify picking up a pallet to simply using an electromagnet. While this simplifies the real-world application of the robot, it still accurately allows us to develop our algorithms and appropriately work on that without distracting us away from designing a mechanical system, tuning it for reliable pickups, and working out any integration issues with it.

5.4 Robot Planner

When it came to the robot planner, there were several approaches that were possible to achieve multi-agent coordination. Ultimately, it was a factor of implementation complexity that was the driving factor behind settling on our final strategy. One potential algorithm would have been to use "Dynamic Planning" [6] in which each robot acts as an independent unit and "re-plans" its path around the other robots if it detects a future collision. While this has the benefit of being robust and "real-time", we found it difficult to determine how to decide which robots get priority and remove the possibility of the robots running endlessly in circles because of contention.

The other approach - which we ended up selecting - is to assign robots priorities and then precompute all of their paths: planning in space and time [14]. In this way, we can ensure that none of the robot paths intersect with each other in space-time configuration space. If a robot has completed its task early, it can drop to the lowest priority and plan around the motion of all the other robots. The only negative to this approach is that the robots must stick to their assigned path. Otherwise, the system is at risk of a collision. We have mitigated this risk by designing a robust controller and including an emergency stop in the form of the user killing the program as a safety measure against collisions.

5.5 Controller

Path following for a mobile robot is typically done with either a Pure Pursuit controller or a PID controller [2]. A Pure Pursuit controller has a lookahead distance, and the robot's target position, or lookahead point, is the point on the path that is the lookahead distance away from the robot. Then, the robot drives in an arc to meet that lookahead point. In a PID controller, the error is defined as the difference between where the robot should be and where the robot currently is. Then, the controller is tuned with gains for the error, the integral of the error, and the derivative of the error and applies a control action according to the sum of these three terms.

The advantage of a Pure Pursuit is that it is fairly simple to implement, and the only tuneable gain is the lookahead distance. However, the issue with a Pure Pursuit controller is that it does not guarantee the orientation of the robot should be aligned with the path that the robot should follow [7]. It only controls the position of the robot over time. This is an issue for our application because we want the robots to be able to pick up and drop off the pallets with a failure rate of under 1%. For the most optimal electromagnet and pallet interface, we want the electromagnet to be directly aligned with the pallet. We also want the pallet to be aligned with the desired drop-off zone. Both of these factors mean that a Pure Pursuit controller is unsuitable for our application.

A PID controller is one of the most ubiquitous types of controllers. It does not need to know the dynamics of the system that it is controlling, and is simple to implement, but takes some time to tune. The disadvantage of this system is that the robot is always trying to catch up to where it should be at the current time [10]. In our application, we are aiming for 0 collisions, which means that the robots need to be following the path to within 5mm at each point in time. This means that the controller also needs to be aware of where the robot should be in the future to know how to control the robot at the current time, which makes a basic PID controller unusable for our application.

We adopted a mix of these two approaches, where we factor in both how the robot should move at the current timestep to reach its target position and orientation at the next timestep as well as how the robot should correct for any differences between its current and expected position and orientation. This approach, with a feedforward controller for how to reach the next waypoint and a feedback controller for reaching the current waypoint, is how mobile robot courses at CMU are taught [10], and we have experience creating and tuning these types of controllers.

6 SYSTEM IMPLEMENTATION

Our main subsystems are the Robots, the Computer Vision section, the Robot Planner, and the Controller. The robots include hardware as well as firmware design, while the rest of the components are all software running on the main computer.

The hardware design for the robots can be found at https://github.com/PARROT-Capstone/Robot-Design.

The firmware for the robots can be found at https://github.com/PARROT-Capstone/Robot-Firmware.

The main computer software can be found at https://github.com/PARROT-Capstone/Robot-Controller.

6.1 Robots

Our robots are constructed out of a custom printed circuit board (PCB) that interfaces between the different subsystems. The schematic for the PCB can be seen in Fig. 15. Fig. 4 contains the labeled CAD for a robot, and Fig. 5 shows a completed robot with the major components labeled.

We chose ESP8266s as our microcontroller (MCU) because they are WiFi compatible and have an antenna built in for communication with the main computer. They use Arduino-style programming, have a USB programmer, and have built-in 3.3V power regulation. All of the I/O connections on the PCB run to the MCU. The MCU runs a pseudo-RTOS (real-time operating system) built on top of the standard Arduino programming paradigm. We have a task for periodically outputting to the screen, a task for listening for commands from the main computer, a task for periodically blinking a heartbeat LED, a task for measuring the voltage of the batteries, a task for updating the Neopixel LED colors, and a task for commanding the drivetrain servos. The screen was used for debugging during initial firmware bring-up, but was taken off once we switched to using fiducial markers instead of Neopixels, as the screen extended too high off the PCB into the fiducial.

Each task has a initialize within the `setup()` method. Within the `loop()` method, each task is called and decides if it needs to do any computation, based either on data updating or the task period expiring.

6.1.1 Batteries and Power Regulation

We are using 2x Samsung 30Q 18650 batteries in series for their high energy density, low DCIR, high discharge current, and easy availability through Carnegie Mellon Racing. These cells will provide an output voltage between 6.0V and 8.4V, depending on their SOC. To regulate this voltage down to a consistent 5V for the rest of the robot, we use a switching converter on the output of the batteries. We needed to add extra capacitance to our 5V rail to prevent our electromagnet from drawing power away from our MCU when it was turned on. We have a voltage divider with two resistors (560 kΩ and 300 kΩ) that converts the possible 8.4V battery voltage to under 3.3V for the MCU to use its internal analog-to-digital converter (ADC) to sense the voltage. In the voltage sense task, the MCU samples the ADC value every second and uses the known resistor values to convert the ADC value back to the voltage that the MCU thinks the batteries are at. The voltage sense task then calls `screenDisplayData()` to alert the screen task that there is new data for the battery voltage to display.

6.1.2 Web Server

The web server uses the `ESP8266WebServer` Arduino Library to listen for POST requests with instructions from the main computer. During the init for the web server task, the task connects to the CMU-DEVICE WiFi network (the MAC addresses for the ESP8266s are given while flashing the MCUs using the Arduino IDE, and they are whitelisted on the CMU WiFi device registration page). The init also

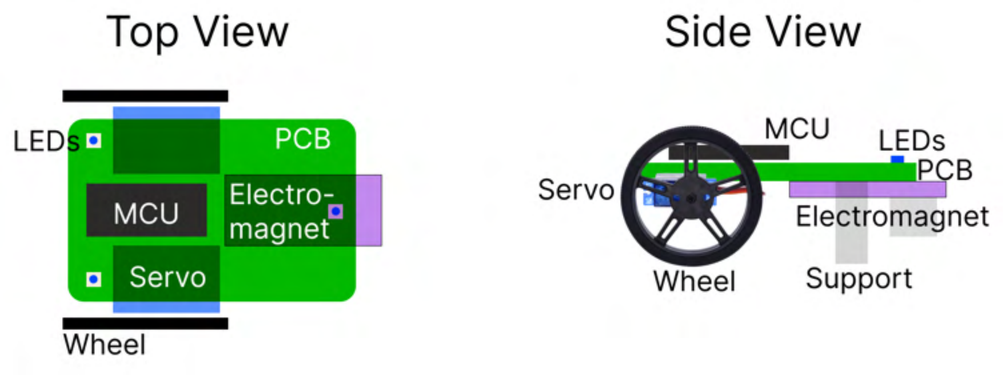


Figure 4: This is the labeled Crayon-CAD for a robot. The top view is on the left, and the side view is on the right.

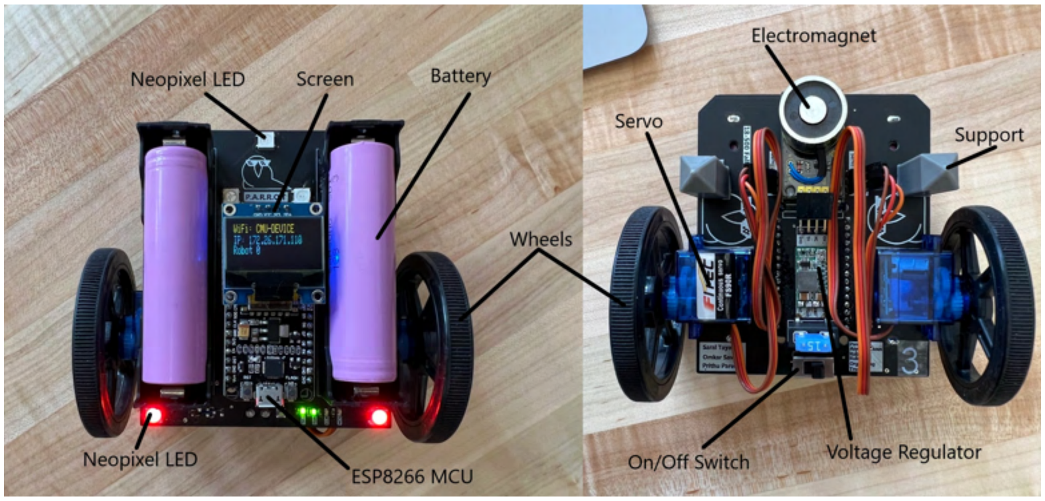


Figure 5: This is the labeled picture of a completed robot. The top view is on the left, and the bottom view is on the right.

starts the server and calls `screenDisplayData()` to alert the screen task that the MCU was able to connect to the WiFi network so that the IP address can be displayed on the screen. When the web server gets a POST request from the main computer, an interrupt is generated and the custom callback handler parses the JSON packet in the POST request. In the handler, if the "dtype", or data type of the packet, indicates that the packet contains servo commands, the handler calls the `drivetrainSetSpeed()` function to set the servo speeds. If the "dtype", indicates that the packet contains an electromagnet command, the handler writes a logic high or low to the electromagnet pin, depending on if the electromagnet should turn on or off. If the "dtype" indicates that the packet should change the Neopixel color, the handler calls the `neopixelSetRobotNumber()` function to change the robot number. The robot number is correlated to the Neopixel LED color within the Neopixel task.

6.1.3 Servos

We are using 9g continuous rotation servos as our drivetrain. Using a continuous rotation servo poses many advantages such as offloading the torque control loop to maintain a velocity target to the servo's internal controller. The continuous rotation servos are also much lighter than standard motors and do not require additional circuitry to drive. These servos have a stall torque of 1.3 kg·cm, and with 3cm radius wheels, each wheel can move 0.433 kg, so the maximum mass our robots can move (including the robots themselves) is 0.867 kg, which is 1.91 lbs. The drivetrain task's initialization uses the Arduino Servo Library to configure the MCU pins that the servos are attached to as software PWM outputs. The servos are updated asynchronously by the web server task. The drivetrain task checks that the web server has sent a command recently and will stop the servos if it has been more than 1 second since the MCU has received a servo command from the web server. This is so that if something goes wrong in the main computer and cannot command servos, the robots will stop in place and avoid collisions. In this case, the software stack on the main computer can be restarted if necessary.

6.1.4 Electromagnet

We are using an electromagnet to be able to pick up the pallets. The pallets are made of metal, and a robot will drive over a pallet, actuate the electromagnet to pick up the pallet, drive to the goal position while holding the pallet, and then deactivate the electromagnet to drop the pallet before driving off to its next goal. The electromagnet is activated and deactivated asynchronously from the main pseudo-RTOS tasks in the handler for the web server.

When we tested the pallet pickup with a fully integrated robot, we discovered that the pallets were too heavy for the robot and created enough torque to tip the robot forward and raise at least one wheel off the ground. To combat this issue, we water-jetted steel plates and attached them to the back underside of the robots to act as counterweights against the pallets.

6.1.5 Neopixels

Although we did not use Neopixels for localization, we chose them for our RGB LEDs since each LED has an internal IC. As such, one doesn't need to worry about current control for brightness control, and instead, we just use an SPI-like communication protocol to address these LEDs to the desired color value. There are also Arduino drivers for commanding Neopixels that we utilize to simplify communication with the LEDs. The Neopixel task is still present in the robots as a backup incase the fiducials fail. The task could be modified to show additional debug messages as different LED colors if needed. The task checks if the button on the MCU was pressed and moves to the next robot number in the sequence. Each robot number corresponds to a unique color for the Neopixels to display. The robot number can also change asynchronously via the web server handler. When the robot number is changed, the Neopixels are updated and `screenDisplayData()` is called so that the robot number is displayed on the screen.

6.1.6 Screen

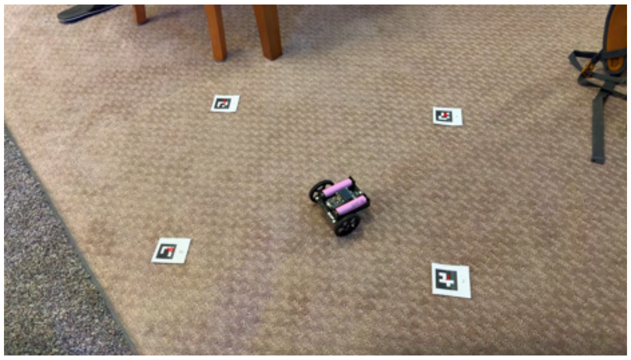
We are using a 128x32 OLED display for initial firmware debugging. The display allows us to show the robot's SOC, the robot's WiFi connectivity status, and the robot's unique ID. The display runs over I2C, and we utilized the Arduino `Adafruit_SSD1306` library to offload graphics processing. The screen displays 4 lines of text, each of which is 8 pixels in height. The screen task initializes the screen and periodically checks if any of the other tasks have `screenDisplayData()`. If the data that the screen needs to display has changed, the screen uses the library functions to issue I2C transactions to clear the screen and write the new data to the screen.

6.2 Computer Vision

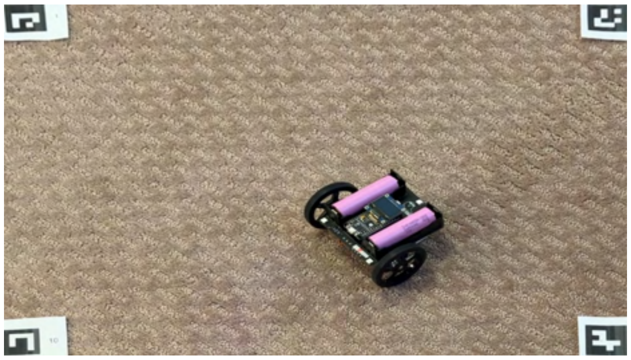
The high-level goal for the computer vision algorithm is to determine where the field is and then determine where the robot is within that field.

Originally our Computer-Vision stack worked by using the principles of feature matching and homography. However, after struggles with getting image masking to work reliably, we decided to switch back to a simpler Arcuo-marker (QR-style) localization system. All of the Arcuo marker localization logic is implemented via OpenCV's helper functions.

As one can see in Fig. 7a, the robot's have a 4x4 style Aruco marker attached on top of them. Similarly the goal and pallets also have similar, yet unique Aruco markers attached as seen in Fig. 8a

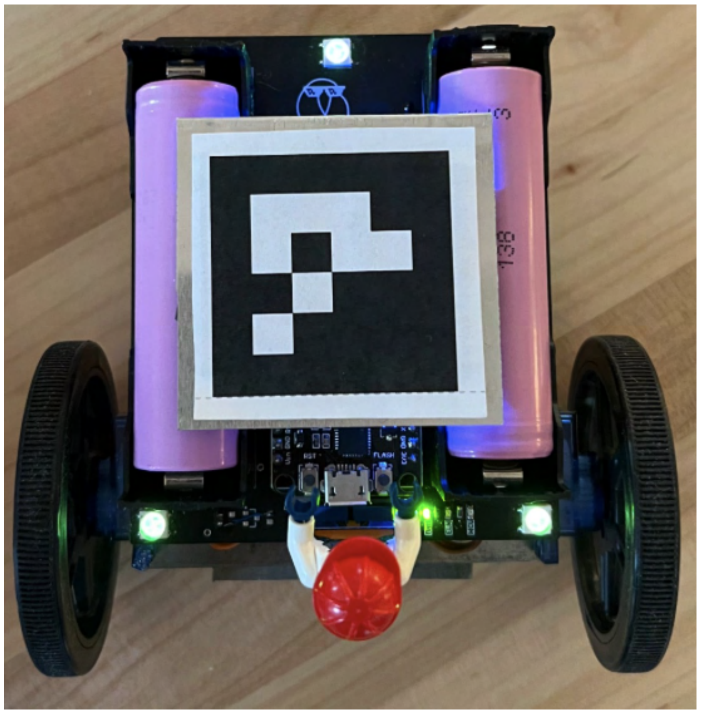


(a) Exaggerated tripod image of sandbox



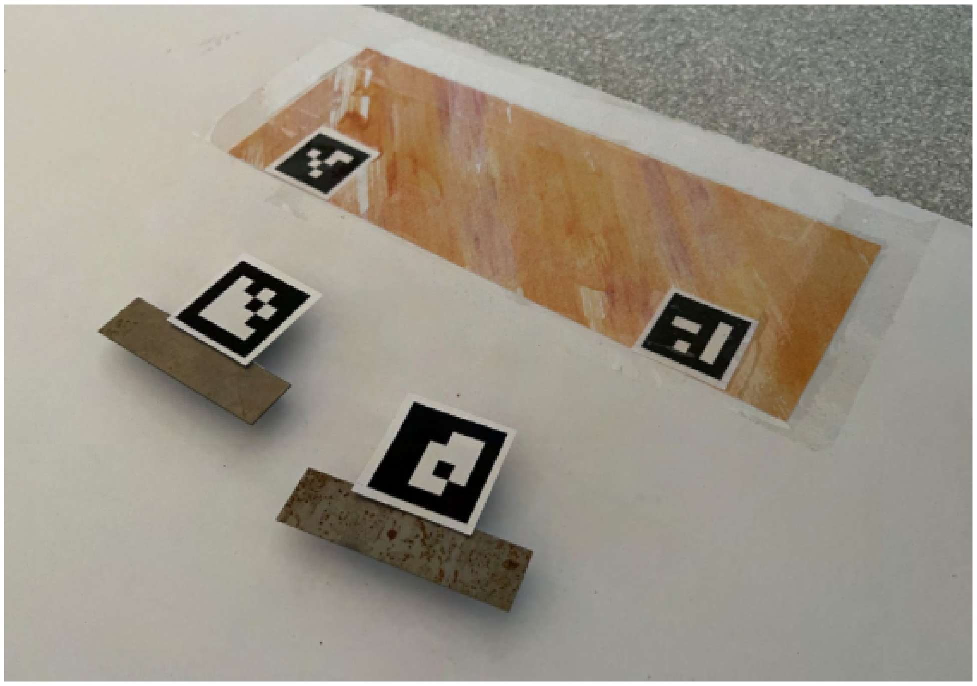
(b) Scaled homography of field

Figure 6: Sandbox detection and calibration



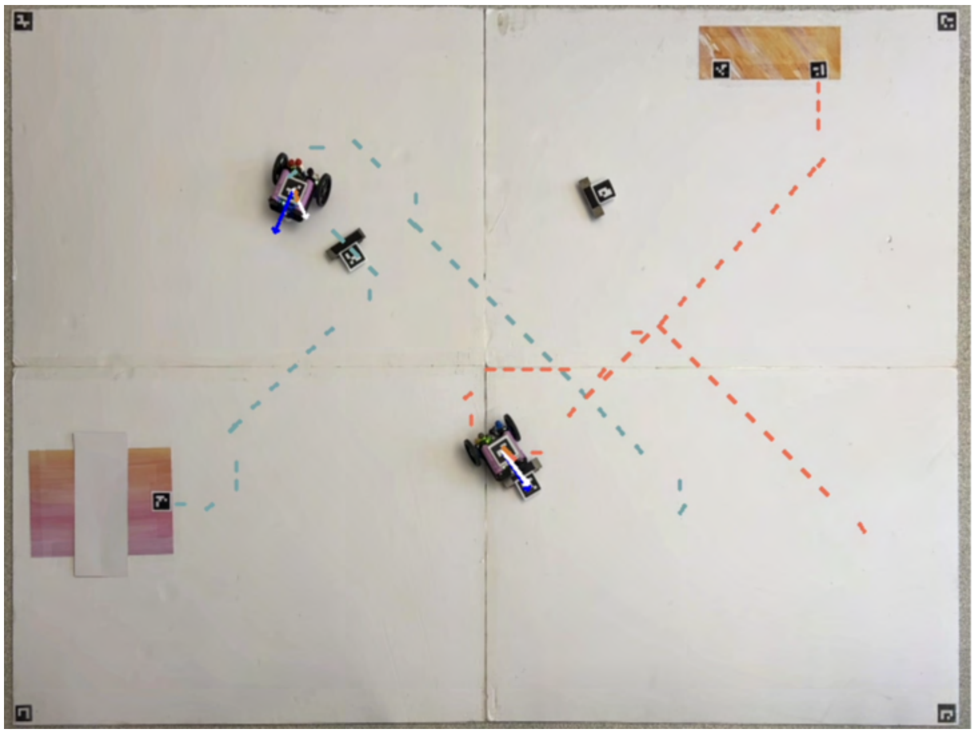
(a)

Figure 7: Robot Top Fiducial



(a)

Figure 8: Pallets and their goal dropoff points



(a)

Figure 9: Sample output from the robot visualizer

6.2.1 Camera Stack

Our Camera hardware was originally an off-the-shelf Logitech C920 Webcam. However, midway through the project, we switched to an IPhone. We did this for 2 reasons. The IPhone camera has a higher dynamic range than the webcam, allowing us to get higher quality video out of the camera. Secondly, the IPhone camera is wireless, allowing us to freely move our main computer anywhere as needed.

The 1080p video is important for us to be able to hit our target localization accuracy. A 1920x1080p image across a 4m x 2m field means each pixel represents 2mm in the real world. Using a lower-resolution video feed would give the algorithm less tolerance for us to hit our 5mm of localization accuracy.

6.2.2 Sandbox Detection

The field detection is done with the use of a fiducial. Fiducials are like QR code tags that are supported by the OpenCV library for easy detection and localization. In particular, the `cv2.aruco` library was used [13].

The computer vision stack is provided with a set of 4 fiducials that will mark the edges of the field as shown in Fig. 6a. When the camera turns on, the system will look for these 4 fiducials and use the pixel positions to find which fiducial is the bottom left, top right, etc.

Once we have the fiducial positions, we can simply use an inverse affine transform (`cv.estimateAffinePartial2D()`) to isolate our sandbox as shown in Fig. 6b!

Lastly, before this sandbox image is returned to the rest of the compute stack, we look at the width of each fiducial detected, and use a hard-coded known real-world width of the fiducial to calibrate our image such that each pixel is one mm in the real world.

6.2.3 Robot Localization and visualizer

The robot localization works on this transformed and flattened sandbox image. Using the flattened image, we can use OpenCv's Aruco localization functions to find the x,y,theta positions of each of the pallets, goals, and robots. All this information is overlayed on a custom visualizer that draws the robot, pallet, and goal poses on the video stream with additional information like the robot paths and feedback + feedforward vectors. One can see this in Fig. 9a

$$\begin{bmatrix} \cos\theta_c(t) & -\sin\theta_c(t) & x \\ \sin\theta_c(t) & \cos\theta_c(t) & y \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

From this transformation matrix, the 3rd column represents the robot's x and y position, and its angle is calculated with (5).

$$\theta = \operatorname{atan2}(\sin\theta_c(t), \cos\theta_c(t)) \tag{5}$$

A visual example of localization can be found in Fig. **??**.

6.3 Robot Planner

The robot planner is divided into two main parts: the task planner and the path planner. The task planner is responsible for assigning the robots to specific pallet-transportation jobs, while the path planner generates the exact motions for the robot to follow to accomplish its specific task. We precompute the tasks and motions for all of the robots and then allow the controller to execute them.

6.3.1 Task Planner

The task planner takes as input a list of pallet+goal pairs plus the positions of all the robots and assigns each robot a pallet and goal task. It prioritizes robots by minimizing the overall distance of the trajectory (i.e. robot to pallet and pallet to goal). The distance used is the straight-line distance between the two sets of points. Overall, the set of assigned tasks should be such that the straight-line distance collectively traveled by all of the robots is minimized.

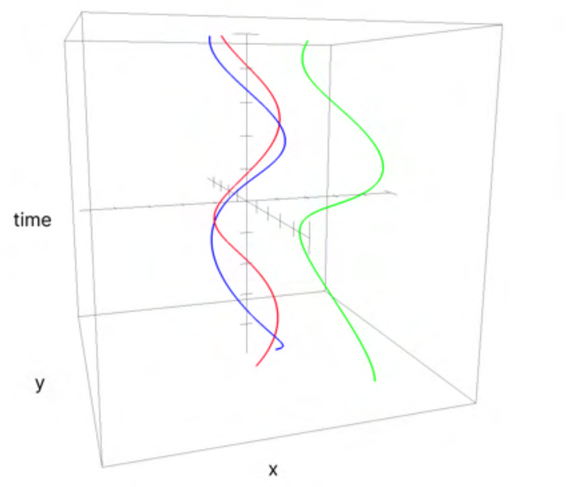


Figure 10: This shows how we are planning in space-time state space. Notice how the paths may overlap in the x,y dimensions but they do not overlap in time. No two robots will be at the same place at the same time.

6.3.2 Path Planner

The path planner is the part of the system that coordinates the motion of all of the robots such that they can collaboratively work without colliding. Each robot has a sub-planner that is responsible for planning its motion given the paths of all other higher-priority robots (as decided by the task planner). The main planner loops over all of the robots in order of priority, planning the path of each of them around static obstacles (i.e. pallets) and the paths of higher-priority robots.

Discretized State Space The state space for our planner is (x, y, θ, t), where t is time. Including time in the state space is important because it allows us to overlapping paths between the multiple robots that do not overlap in time. We have discretized the map into a 2d grid with squares of 10mm each, with 8 different possible angles for the robot at each square. We have generated a set of motion primitives for the robot which respect its non-holonomic turn radius constraints. What this means is that given an (x, y, θ) we define a set of possible next states the robot can be in given a set of small-predefined motions we know the robot is able to make.

Graph Search Each sub-planner plans a path in two segments: from robot to pallet and from pallet to goal. Two different A* graph searches are run: one for each segment. We have modified this algorithm to allow for the time variable to come into play. When a state (x, y, θ) is explored by the algorithm, it is assigned a time that the robot is expected to be at that state based on the previous state's time, the distance between the states, and the robot's velocity. This state is then checked for collisions against all static obstacles as well as the dynamic positions of other robots at the state's assigned time. If it is in static or dynamic collision, it is ignored. Note that it is possible for the (x, y, θ) to be explored by A* several instances with multiple assigned times. This allows the robot paths to intersect in physical space while not colliding since their physical path intersection happens at different points in time.

Collision Checking For static collision checking (i.e. against non-moving objects) we simply check for an intersection between the robot's footprint and any obstacles. For dynamic collisions, we loop through all the paths of higher-priority robots and check for overlapping footprints of robots at the same timestep with a buffer of 5 seconds. This buffer prevents minor errors in the control system from resulting in a collision by keeping robots far enough apart from each other.

6.4 Control System

The control system takes the paths generated by the motion planner and the current positions for each robot given by the computer vision section and generates HTTP POST requests to command the robots' servos and electromagnets such that the robots follow the path as accurately as possible. To do this, each robot has its own individual controller that interpolates between the waypoints in the path, generates the feed-forward and feedback terms in the robot's forward linear velocity and angular velocity, and commands the servo speeds.

6.4.1 Waypoint Interpolation

The path that the robot should follow includes the robot's position, orientation, and time, the controller needs to interpolate between the adjacent waypoints to determine where the robot should be at each point in time between these waypoint times. If the adjacent waypoints have the same robot orientation, (6) describes how the linear interpolation is calculated

$$\begin{cases} x^*(t) &= \frac{x_2-x_1}{t_2-t_1}(t-t_1)+x_1 \\ y^*(t) &= \frac{y_2-y_1}{t_2-t_1}(t-t_1)+y_1 \\ \theta^*(t) &= \theta_1=\theta_2 \end{cases} \tag{6}$$

where $(x_1, y_1, \theta_1, t_1)$ and $(x_2, y_2, \theta_2, t_2)$ are the two adjacent waypoints with $t_2 > t_1$ and $(x^*(t), y^*(t), \theta^*(t))$ is the desired target robot position and orientation for time t with $t_1 \le t \le t_2$.

If the adjacent waypoints do not have the same robot orientation, two cubic hermite splines are calculated, using the same notation as above, for both x^* and y^* as functions of t in the domain $t_1 \le t \le t_2$. Equation (7) describes the relationship between θ and robot's velocity vector. This invariant must be met at each of the waypoints. The `scipy.interpolate.BPoly.from_derivatives` Python function is used to calculate the cubic hermite splines subjected to the constraints in (8) for a given curviness factor, k.

$$\tan\theta = \frac{dy/dt}{dx/dt} \tag{7}$$

$$\begin{cases} x^*(t_1) &= x_1 \\ \left.\frac{dx^*}{dt}\right|_t &= k*\cos\theta_1 \\ x^*(t_2) &= x_2 \\ \left.\frac{dx^*}{dt}\right|_t &= k*\cos\theta_2 \\ y^*(t_1) &= y_1 \\ \left.\frac{dy^*}{dt}\right|_t &= k*\sin\theta_1 \\ y^*(t_2) &= y_2 \\ \left.\frac{dy^*}{dt}\right|_t &= k*\sin\theta_2 \end{cases} \tag{8}$$

The desired orientation of the robot along this cubic interpolation is given by (9).

$$\theta^*(t) = \operatorname{atan2}\left(\left.\frac{dy^*}{dt}\right|_t, \left.\frac{dx^*}{dt}\right|_t\right) \tag{9}$$

6.4.2 Feed Forward Control

The feedforward control is the component of the control action that assumes the robot is always on track with the target pose specified by the interpolation. It is trying to move the robot to the next timestep on the interpolation [3]. Equation (10) shows how to calculate the feedforward terms for linear velocity and angular velocity. The linear velocity feedforward term is the magnitude of the velocity vector for the interpolated path in the x and y directions. The angular velocity feedforward term is computed as the product of the linear velocity feedforward term and the curvature of the interpolated path in the x and y directions.

$$\begin{cases} v_{ff}(t) &= \sqrt{\left(\frac{dy^*}{dt}\Big|_t\right)^2 + \left(\frac{dx^*}{dt}\Big|_t\right)^2} \\ \omega_{ff}(t) &= \frac{\frac{dx^*}{dt}\big|_t \cdot \frac{d^2y^*}{dt^2}\big|_t - \frac{dy^*}{dt}\big|_t \cdot \frac{d^2x^*}{dt^2}\big|_t}{(v_{ff}(t))^2} \end{cases} \tag{10}$$

The Python function `scipy.misc.derivative` is used to calculate the first and second derivatives of the interpolated path between waypoints.

6.4.3 Feedback Control

The feedback control is the component of the control action that is trying to correct for errors between where the robot currently is and where it should be. For current pose $(x_c(t), y_c(t), \theta_c(t))$ and desired pose $(x^*(t), y^*(t), \theta^*(t))$, (11) describes the error $(x_e(t), y_e(t), \theta_e(t))$ with reference to the robot's coordinates.

$$\begin{bmatrix} x_e(t) \\ y_e(t) \\ \theta_e(t) \end{bmatrix} = \begin{bmatrix} \cos\theta_c(t) & \sin\theta_c(t) & 0 \\ -\sin\theta_c(t) & \cos\theta_c(t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x^*(t) - x_c(t) \\ y^*(t) - y_c(t) \\ \theta^*(t) - \theta_c(t) \end{bmatrix} \tag{11}$$

Then, the linear velocity and angular velocity feedback terms are given by (12) where the K_P, K_I, and K_D gains are tune-able using standard PID tuning methods [10].

$$\begin{cases} v_{fbk}(t) = & K_{Px} \cdot x_e(t) + K_{Ix} \cdot \int x_e(t)\mathrm{d}t + K_{Dx} \cdot \frac{\mathrm{d}x_e(t)}{\mathrm{d}t} \\ \omega_{fbk}(t) = & K_{Py} \cdot y_e(t) + K_{Iy} \cdot \int y_e(t)\mathrm{d}t + K_{Dy} \cdot \frac{\mathrm{d}y_e(t)}{\mathrm{d}t} \\ & + K_{P\theta} \cdot \theta_e(t) + K_{I\theta} \cdot \int \theta_e(t)\mathrm{d}t + K_{D\theta} \cdot \frac{\mathrm{d}\theta_e(t)}{\mathrm{d}t} \end{cases} \tag{12}$$

6.4.4 Robot Commands

Once the feedforward and feedback terms are calculated, (13) describes how to compute the control action, where $v(t)$ represents the linear velocity to command to the robot and $\omega(t)$ represents the angular velocity to command to the robot.

$$\begin{cases} v(t) &= v_{ff}(t) + v_{fbk}(t) \\ \omega(t) &= \omega_{ff}(t) + \omega_{fbk}(t) \end{cases} \tag{13}$$

Then, one can use (14) to calculate the linear velocities of each of the wheels, where $v_l(t)$ represents the linear velocity of the left wheel, $v_r(t)$ represents the linear velocity of the right wheel, and b is the wheelbase (the distance between the points of contact between the wheels and the ground).

$$\begin{cases} v_l(t) &= v(t) - \omega(t) * \frac{b}{2} \\ v_r(t) &= v(t) + \omega(t) * \frac{b}{2} \end{cases} \tag{14}$$

Finally, the servo PWM duty cycles must be calculated. Equation (15) shows how to convert the linear velocity of the wheels to servo duty cycles. The servo duty cycle ranges from 0 to 180. 0 represents the wheel rotating counter-clockwise, and 180 represents the wheel rotating clockwise. Since the right servo is mounted backward on the robot and a duty cycle of 90 corresponds to the wheel being at rest, the right servo duty cycle is inverted, and both duty cycles are offset by 90. r represents the radius of the wheel, and K_ω represents the slope of the relationship between desired wheel angular velocity and duty cycle.

$$\begin{cases} S_l(t) &= 90 + \frac{v_l(t)}{r} \cdot K_\omega \\ S_r(t) &= 90 - \frac{v_r(t)}{r} \cdot K_\omega \end{cases} \tag{15}$$

Once the servo PWM duty cycles are computed, the controller can send an HTTP POST request with the packet data type indicating that the packet contains servo commands and the rest of the packet containing the left and right servo duty cycles.

Each waypoint also has a tag associated with the position, orientation, and time. This tag indicates whether the robot should be driving, picking up the pallet, or dropping off the pallet. If the tag associated with the current waypoint indicates that the robot should pick up the pallet, the controller sends an HTTP POST request to the robot with the instructions in the packet for actuating the electromagnet. If the tag associated with the current waypoint indicates that the robot should drop off the pallet, the controller sends an HTTP POST request to the robot with the instructions in the packet for de-actuating the electromagnet. Otherwise, the controller does not send any instructions regarding the electromagnet to the robot.

6.4.5 Emergency Stop

In the case when we may need to stop all robots from the User Interface, either clicking the emergency stop button or hitting Ctrl-C in the terminal running the main computer program will cause all controllers to send an HTTP POST request to their respective robots with both wheel velocities equal to 0, which corresponds to a duty cycle of 90. This will cause all robots to stop moving.

7 TEST, VERIFICATION, & VALIDATION

Outlined below are the various tests we ran to validate our design goals. We have also compiled a table mapping metric to test and results for easy reference in Table 1.

7.1 Tests for power consumption and battery life

We ran a repeated loop of robot pickup → move → robot dropoff → move → pause. This was a fairly accurate representation of our robot's average action while letting us run a continuous loop. We timed the robots from a full charge of 4.2V per cell and measured the cells after 3 hours. The average battery cell across the 3 robots had a voltage measurement of 3.57V per cell, which corresponds to about 50% SOC of a 18650 Samsung 30Q cell. Extrapolating this to see how long the robots would have lasted on a full charge results in a battery life of 6 hours. We previously had issues with the electromagnet discharging the 5V power rail when it was turned on, causing the entire robot to reset momentarily. However, adding a large capacitor between the 5V power rail and the ground aided in preventing our robots from resetting and extended our battery life beyond our desired 4 hours of runtime.

7.2 Computer Vision Resolution Tests

We placed the robots and pallets in 10 arbitrary but fixed positions for 5 repetitions. We measured the accuracy of the localization against ground truth measurements. We aimed for localization accuracy of less than 5mm and achieved this metric with an average accuracy of 0.8mm.

7.3 Latency Tests

We gave the robots 20 start-goal pairs and timed the elapsed time from each iteration sense-plan-act software loop. We averaged the elapsed loop time to get an understanding of latency across 10 minutes of operation of 3 robots. Initially, as seen in Fig. 11, we had a latency of 115ms with one robot, as we performed our entire sense-plan-act loop sequentially. We were aiming for a latency of 200ms, and this would have met this metric, but we had to run our robots at a speed of around 1.1 cm/s to ensure that the robots could stay on their path with this latency. This was much slower than we would have liked, as the robots inched across the field along their paths. This sequential software did not scale well to multiple robots, especially since most of the bottleneck was transmitting the commands to the robots over WiFi POST requests. To reduce the loop latency, we created a separate thread for each robot just to send the command to the robot. The main thread would capture an image and run the localization and controllers, while the communication threads would take the last output from the controllers and send the left and right wheel speeds to the robots. From there, we noticed that we could parallelize this further by splitting capturing an image into a separate thread that sends data to the main thread for localization. This brought our loop latency down to 35ms, and that latency barely increases as we add more robots, as shown in Fig. 11.

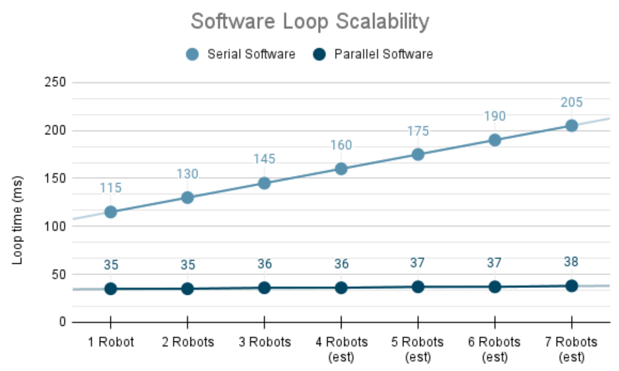


Figure 11: Graph of sense-plan-act loop for increasing number of robots

7.4 Controller & Actuator Tests

We gave the robots 5 different curvature start-goal pairs and measured the accuracy and precision of the robots' execution of the paths across 5 runs. We were aiming for a controller accuracy of under 5mm, and our maximum average controller error across the 25 runs was 2.3mm. See Fig. 12. It took many iterations of tuning our control algorithm and 3 sets of PID values to achieve this. Our controller and localization system was robust enough to rejoin its desired path even when there was adversarial noise injected into the system by moving the robot off of its path or holding the robot back.

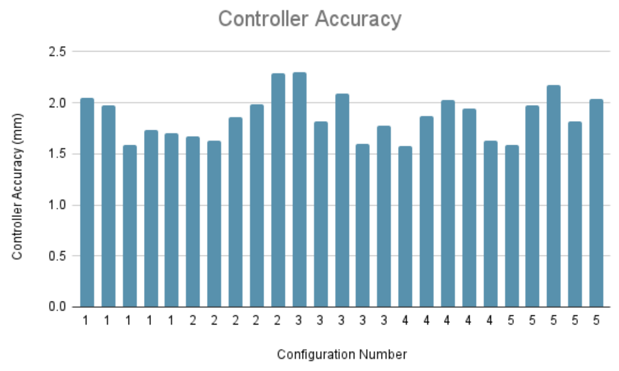


Figure 12: Bar graph of average controller error for different configurations

7.5 Planner Tests

We planned 50 layouts with various numbers of robots, either 1, 2, or 3, and ensured that no paths collided in both space and time and followed the minimum clearance distance between robots. We achieved this metric but noticed that around 1 in 5 times, the planner fails to plan paths for all robots on the field, often due to the positioning of the robots and pallets and a high likelihood of potential robot-robot collisions. In these sub-optimal situations, the planner gives the paths for as many robots as it can, and those robots that don't have a path are staying in place until the moving robots have reached their goals. And once the moving robots have dropped off their respective pallets, the robot planner recomputes any paths so that the remaining pallets can be picked up. There were no robot-robot collisions in any of these tests, but there were collisions in 6% of these tests between a robot that had picked up a pallet and a different pallet on the field. This was likely due to the planner approximating the robot as a circle with a given radius, and that radius inflated once the robot picked up a pallet. Although this approximation speeds up the collision-checking software, there is a potential for unintended collisions between robots and pallets, since neither of them is a perfect circle.

7.6 Scalability Testing

We planned and executed 2 arbitrary, but fixed, map layouts across a sweep of robot counts from 1 to 3. We timed the execution time loading all the 'trucks' and compared the average speedup given the independent variable of robots in the field. Fig. 13 shows the speedup graph for one of these trials. In both trials, we noticed that we achieved near-perfect speedup at 3 robots (3x speedup at 3 robots). This met our overall goal of 2x speedup at 3 robots. Our result was because with 1 robot, that robot has to do extra work to move from one goal to pick up its next pallet, resulting in a larger total traveling distance for 1 robot than 3 robots. The knee in the speedup graph at 2 robots was because with 3 pallets, the first two pallets can be dropped off at the same time, and then one robot is idle, as only one robot can pick up the last pallet.

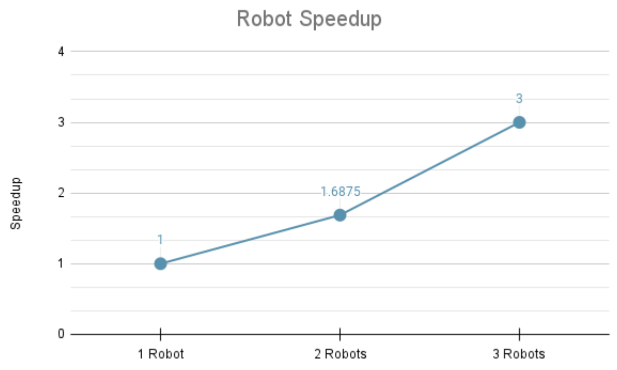


Figure 13: Graph of speedup for increasing number of robots

7.7 Pickup and Dropoff Testing

We ran our system 50 times against different robot-pallet-goal tuples and measured pickup and dropoff reliability. Our goal was a >99% pickup rate and a dropoff distance of 5mm. In our testing, we achieved a 90% pick-up rate and 84% drop-off rate. This failure can be attributed to the robot being slightly misaligned in orientation when it tries to pick up the pallet. Most of the time, the system is able to correct this, but sometimes the pallet gets knocked sideways and lines up with the robot's wheel rather than the electromagnet, resulting in the robot driving over the pallet. The proper fix for this would be to have a dynamic planner around the pallet to ensure that the robot has picked up the pallet and retry pickup if the robot has not picked it up the first time. The dropoff issue was partially caused by the pallet getting magnetized by the electromagnet if the robot has held onto the pallet for a long time. This was mitigated by adding a piece of electrical tape on the pallet to increase the distance between the pallet and the electromagnet, slightly decreasing the magnetic field. The other issue during dropoff was that the robot was over 5mm away from the desired dropoff location. A fix for this would be to allow the controller to have more time to correct for errors before the path planner says to drop off the pallet at that time step.

8 PROJECT MANAGEMENT

8.1 Schedule

Given the ambitious nature of our project which involves custom hardware, complex software systems, and sensor integration, we had to start quite early and aim for an aggressive timeline. Our Gantt chart is viewable in Fig. 14.

8.2 Team Member Responsibilities

Each team member's unique responsibilities are listed below:

- Saral was responsible for the hardware design of the robot and the computer vision software.
- Prithu was responsible for the task planning and motion planning software.
- Omkar was responsible for the robot's firmware and the controller software.

Given the complex, interdisciplinary nature of the project, we wanted to make sure that the initial design and integration of the project were done by all members of the team to ensure the minimization of specifications, requirements, and system weaknesses from falling through the cracks.

8.3 Bill of Materials and Budget

The bill of materials and budget per robot can be found in Table 2.

The primary expense of this project is the Robot BOM. The only additional hardware to the robot is a camera and a personal computer. A webcam was borrowed from the ECE 18-500 Inventory, but we preferred to use an iPhone camera.

Table 1: Testing Summary

Requirement	Metric	Test	Result
Localization Accuracy	<5mm	Place robot and pallets in 10 arbitrary, but fixed positions for 5 cycles. Measure accuracy and precision of localization system	0.8mm
Controller Accuracy	<5mm	Have robot execute 20 unique start-goal waypoint pairs 5 times. Measure position accuracy.	2.3mm
Path Planning Collision avoidance	100%	Plan 50 layouts with 3 or more robots and ensure no paths collide	0 robot-robot collisions, but robot-pallet collisions occur 6% of the time
Scalability testing	2x across 3 robots.	Run system against 15 arbitrary, but fixed map layouts with a varying number of robots	3x at 3 robots
Battery Runtime	>4 hours	Run test by having robots run a loop of tasks till the battery dies.	6 hours
Pickup and Dropoff	>99% reliability on both pickup and dropoff	Run system 50 times against different robot-pallet-goal tuples and record pickup and dropoff success rate	90% pickup; 84% dropoff

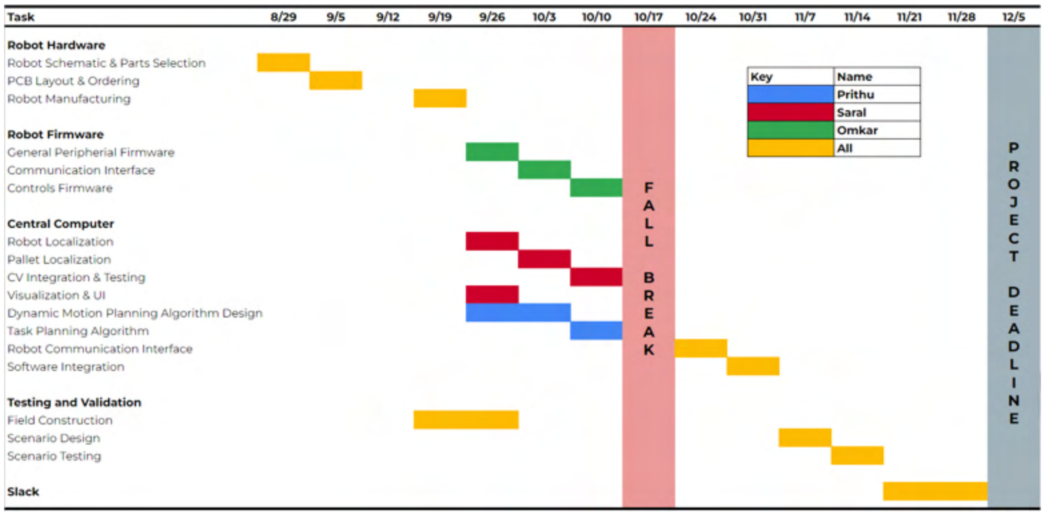


Figure 14: Gantt Chart

Table 2: Bill of materials per Robot

Description #	Manufacturer	Quantity	Cost @	Total
9g Servos with wheels	Geekstory	2	$6.25	$13.50
NodeMCU	KeeYees	1	$4.50	$4.50
PCB	PCBway	1	$6.17	$6.17
NeoPixels	NA	5	$0.09	$0.45
Electrical Passives	NA	1	$5	$5
OLED Screen	Hosyond	1	$3	$3
18650 Cells	Samsung	2	$5	$10
18650 CellHolders	abcGoodefg	2	$0.9	$1.8
Electromagnet	KeyStudio	1	$10	$10
Switch	Cylewet	1	$0.04	$0.04
Fuse	NA	1	$2	$2
Power Regulator	Drok	1	$2	$2
Misc headers	NA	1	$2	$2
				$60.46

8.4 Risk Mitigation

Early in the semester, we completed a preliminary risk evaluation and risk mitigation and produced many MVP demos that proved the viability given our initial risks of project complexity and timeline issues. The following list of risks and risk mitigations are based on the outstanding work given our current position.

8.4.1 Webcam sensitivities

We have noticed that the webcam was quite sensitive to the environment in which it is operating. Simple changes in ambient lighting (room getting brighter/dimmer, like a cloud outside passing over the sun) affected the camera's auto-exposure, auto-whitebalance, and dynamic range. This makes our system quite sensitive and not optimal.

As risk mitigation, we found a way to manually control our camera's settings and disable auto-exposure and auto-white balance features. Furthermore, we were experimenting with a dynamic calibration method to figure out what robots are in the environment rather than hard-coded RGB values for each robot using a Look-Up Table, but we pivoted away from Neopixesl instead of fiducial markers for localization, rendering this algorithm obsolete. More information about our approaches can be found in Section 6.2.3.

This proved to be unstable, especially with the camera's auto-focus causing the computer vision system to intermittently lose track of the robots. Therefore, we switched to using our iPhones as our webcam due to their exceptional dynamic range capabilities. We had already performed a quick demo to analyze the phone to computer wireless latency and found it to be under 20-30 ms. This was more than acceptable for our needs. If anything, this may be a better approach if we intend to launch our project as an open-source robotic platform as more people have access to personal smartphones than people have access to $100+ USB webcams.

8.4.2 Timing issues

Halfway through the semester, most of our demos used only a subset of all the systems that were needed in the whole robot stack. We had a hard-coded emulation to simulate the other systems. For example, we tested the robot controller using our computer vision localizer but with simulated paths from the motion planner. Similarly, we have tested the motion planner in isolation.

We were concerned that integrating everything would affect the timing and latency of the entire system making it unstable and not working as expected. To mitigate this risk, we have pulled our integration timeline up and allocated more time to iron out potential issues like these. In the end, we parallelized our software stack so that one thread was polling for images from the camera, another thread did the localization and control for all the robots, and one thread per robot was communicating the controllers' output to the individual robots.

9 ETHICAL ISSUES

While the ethical issues for this project are minimal given that the system is not used by the general public, there are still several important factors to consider.

9.1 Displacement of Workers

As with all technologies that seek to automate work previously done manually by humans, the implementation of our system will most likely result in the displacement of workers who currently work to move items around warehouse floors. This shift will occur in large part due to the large efficiency gains (both in terms of time and financial cost) that our system will provide over the current status quo. The solution to this issue is not to prevent the implementation of our system. If we don't proceed with this product, it is highly likely that another group/company will. Instead, a better approach is to find ways to retrain the workers so that they either have the skills necessary to help manage and maintain this new system or have the ability to take on a different job that has not yet been automated.

9.2 Security & Data Privacy

Another ethical risk that our system faces is security and data privacy. Since our system is fully wireless and Wi-Fi-based, it is prone to be hacked and taken over by malicious actors. One way to mitigate this risk is to ensure that the system is on a network either not connected to the outside internet or heavily protected by firewalls. The system should also have emergency-stop and safety systems in place that allow for an immediate system override, should an outside actor gain control of the robots.

Another risk surrounds data privacy. Our system uses a camera as the main component of our computer vision stack and is used to localize the robots, pallets, and goal positions. In the real world, it is possible that this camera can pick up images of warehouse employees and pose privacy concerns. To address this issue we have made it such that our system doesn't store any image taken by the camera. These images are discarded immediately after localization.

9.3 Safety of Warehouse Employees

The biggest ethical issue that we face is warehouse employee safety. In the real-world, it is extremely likely that there will be people working in warehouses alongside our robots. In this situation, it is highly important that our robots behave in a safe manner that doesn't collide with or otherwise harm these humans. In our current state, our robots are not able to account for uncertain dynamic obstacles (i.e. humans) in their environment. This is an area of future work discussed in Section 11.1 that must be worked on for our system to be safe enough to work with humans. In addition to this, we must also add sensors to the robot such that they can emergency stop if they detect they are too close to obstacles in their environment not detected by the CV system.

10 RELATED WORK

There are several different systems being developed to tackle the issue of warehouse automation. The one most similar to ours is the space of Automated Guided Vehicles [16]. These types of systems use robot forklifts to move pallets around a warehouse floor and into trucks. Another type of system is Automated Storage and Retrieval systems which allow for goods on the warehouse floor to be brought to humans quickly [16]. An example of this is the Amazon Kiva robots [8]. Finally, we have pick-and-place robots which are more used for organizing products on an assembly line [16]. These robots are used in a warehouse setting, but not for the same task we are trying to solve.

11 SUMMARY

In this project, we recreated the parallelization of robots in a factory/warehouse setting via a tabletop model. By using custom, modular robotics with the PCB as the frame, we were able to design and build extremely simple and mass-producible robots. Using computer vision for localization of the robots, pallets, and trucks, we created a path within a (x, y, angle, time) state space and enforced that path with a controller to eliminate manufacturing errors between robots. Our goal was a 2x speedup with 3 robots which we exceeded with a 3x speedup for 3 robots.

11.1 Future Work

There are three main areas for future work: communication protocol updates, dynamic planning, and servo improvements.

11.2 Communication Protocol Updates

One of the biggest issues that we faced with the robustness of our system was the inherent latency of CMU's Wi-Fi network. This had major effects on the quality of our controller as slow or dropped packets made it increasingly hard for the robot to reliably and accurately follow the path it was provided. As an improvement on this, we would like to test with our own private network to see if it can provide any improvements over the current system. We suppose it should be better given the significant reduction in network traffic and lack of firewalls. Furthermore, we would like to experiment with other network protocols (ex. Bluetooth / Zigbee) to see if they provide any further gains over Wi-Fi.

11.3 Dynamic Planning

Given the scope of this class, we decided to have our planner precompute all of the robot paths that would then be executed by the controller. While this planner works well for our current purposes, it is not able to react to uncertainties in the environment (ex. unsuccessful pallet pickup, humans). For these situations, it would be beneficial to have a "dynamic" planner which can keep checking the state of the environment and quickly replan parts of the path if it is detected the robot is on a collision course or performed unexpectedly. These types of "dynamic" planning are commonly used in current industrial applications.

11.4 Servo Improvements

The current servos used on our robots are cheap, yet extremely unreliable leading to a significant burden on our controller feedback loop to correct for the inherent errors and make our robot follow its desired path. Purchasing more accurate servos will enable our robots to travel faster with increased accuracy.

11.5 Lessons Learned

If future groups want to explore projects in the area of swarm robotics and multi-agent path planning and control, they should be aware of the work that has to happen before any of the subsystems could be integrated. We wrote simulators to test the motion planner and the controllers to ensure that they were working as expected before we could integrate them with the computer vision subsystem. We also had to validate the localization algorithm was working independently before we could feed that into the planner and the controller. Then, when it came time to integrate with the hardware, we spent the majority of a weekend trying to get a single robot to drive in a straight line. There were issues with the coordinate frames that were difficult to hunt down. The localizer used the image coordinate frame and returned an angle between 0 and 2π, but in the clockwise direction. The planner used a Cartesian coordinate frame with counterclockwise angles between 0 and 2π. The controller was also using a Cartesian coordinate frame but with counterclockwise angles between $-\pi$ and π. Resolving this issue became a quite difficult challenge for us.

There were several hardware bugs that we discovered during integration, like the electromagnet drawing too much current and causing our regulator to reset our MCU, or the pallets getting magnetized once the robots have held onto them for too long. We were able to fix these issues, but it was integral that we had prior experience debugging hardware as well as software to understand how to diagnose and fix these issues.

Glossary of Acronyms

- ADC - Analog to Digital Converter

- BOM - Bill of Material
- CAD - Computer Aided Design
- DOF - Degree of freedom
- DCIR - Direct Circuit Internal Resistance
- HSV - Hue, Saturation, Vibrance
- HTTP - Hypertext Transfer Protocol
- IC - Integrated Circuit
- JSON - JavaScript Object Notation
- MAC - Media Access Control
- MCU - Microcontroller Unit
- PID - Proportional, Integral, Derivative control
- PCB - Printed Circuit Board
- PWM - Pulse Width Modulation
- SOC - State of charge

References

[1] *Amazon Robotics Uses Amazon SageMaker and AWS Inferentia to Enable ML Inferencing at Scale.* 2022. URL: `https://aws.amazon.com/solutions/case-studies/amazon-robotics-case-study/`.

[2] Omead Amidi. *Integrated Mobile Robot Control.* Tech. rep. CMU-RI-TR-90-17. Pittsburgh, PA: Carnegie Mellon University, 1990.

[3] Muhammad Asif et al. "Feedforward and feedback kinematics controller for wheeled mobile robot trajectory tracking". In: *Journal of Automation and Control Engineering* 3.3 (2015), 178–182. DOI: `10.12720/joace.3.3.178-182`.

[4] David Benady. *How robotics are Optimising Warehouse Operations.* 2017. URL: `https://www.raconteur.net/technology/automation/how-robotics-are-optimising-warehouse-operations/`.

[5] Chris Cacioppo. *Advantages of system-directed robots vs. swarming robots.* 2022. URL: `https://6river.com/what-are-the-advantages-of-directed-picking-robots-vs-swarming-robots/`.

[6] Devin Connell and Hung Manh La. *Dynamic Path Planning and Replanning for Mobile Robots using RRT*.* Number: arXiv:1704.04585 arXiv:1704.04585 [cs]. Apr. 2017. URL: `http://arxiv.org/abs/1704.04585` (visited on 10/14/2022).

[7] R. Craig Coulter. *Implementation of the Pure Pursuit Path Tracking Algorithm.* Tech. rep. CMU-RI-TR-92-01. Pittsburgh, PA: Carnegie Mellon University, 1992.

[8] *How Kiva Systems and Warehouse Management Systems Interact | RoboticsTomorrow.* en-US. URL: `https://roboticstomorrow.com/article/2011/12/how-kiva-systems-and-warehouse-management-systems-interact/23/` (visited on 10/15/2022).

[9] Marius Jurt et al. "Collective transport of arbitrarily shaped objects using robot swarms". In: *Artificial Life and Robotics* 27.2 (2022), 365–372. DOI: `10.1007/s10015-022-00730-5`.

[10] Alonzo Kelly. *Mobile Robotics: Mathematics, models, and methods.* Cambridge University Press, 2013.

[11] Geekplus International Company Limited. *Robotics.* 2022. URL: `https://www.geekplus.com/technology/robotics`.

[12] *Robots.* 2022. URL: `https://cajarobotics.com/robots/`.

[13] Adrian Rosebrock. *Detecting aruco markers with opencv and python.* 2021. URL: `https://pyimagesearch.com/2020/12/21/detecting-aruco-markers-with-opencv-and-python/`.

[14] Dawei Sun et al. *Multi-agent Motion Planning from Signal Temporal Logic Specifications.* Number: arXiv:2201.05247 arXiv:2201.05247 [cs, eess]. Jan. 2022. URL: `http://arxiv.org/abs/2201.05247` (visited on 10/14/2022).

[15] *Tesla plans 'thousands of humanoid robots within factories'.* 2022. URL: `https://electrek.co/2022/09/23/tesla-thousands-humanoid-robots-factories/`.

[16] *Warehouse Robotics & Different Types of Robots.* en-US. Jan. 2020. URL: `https://6river.com/what-is-warehouse-robotics/` (visited on 10/14/2022).

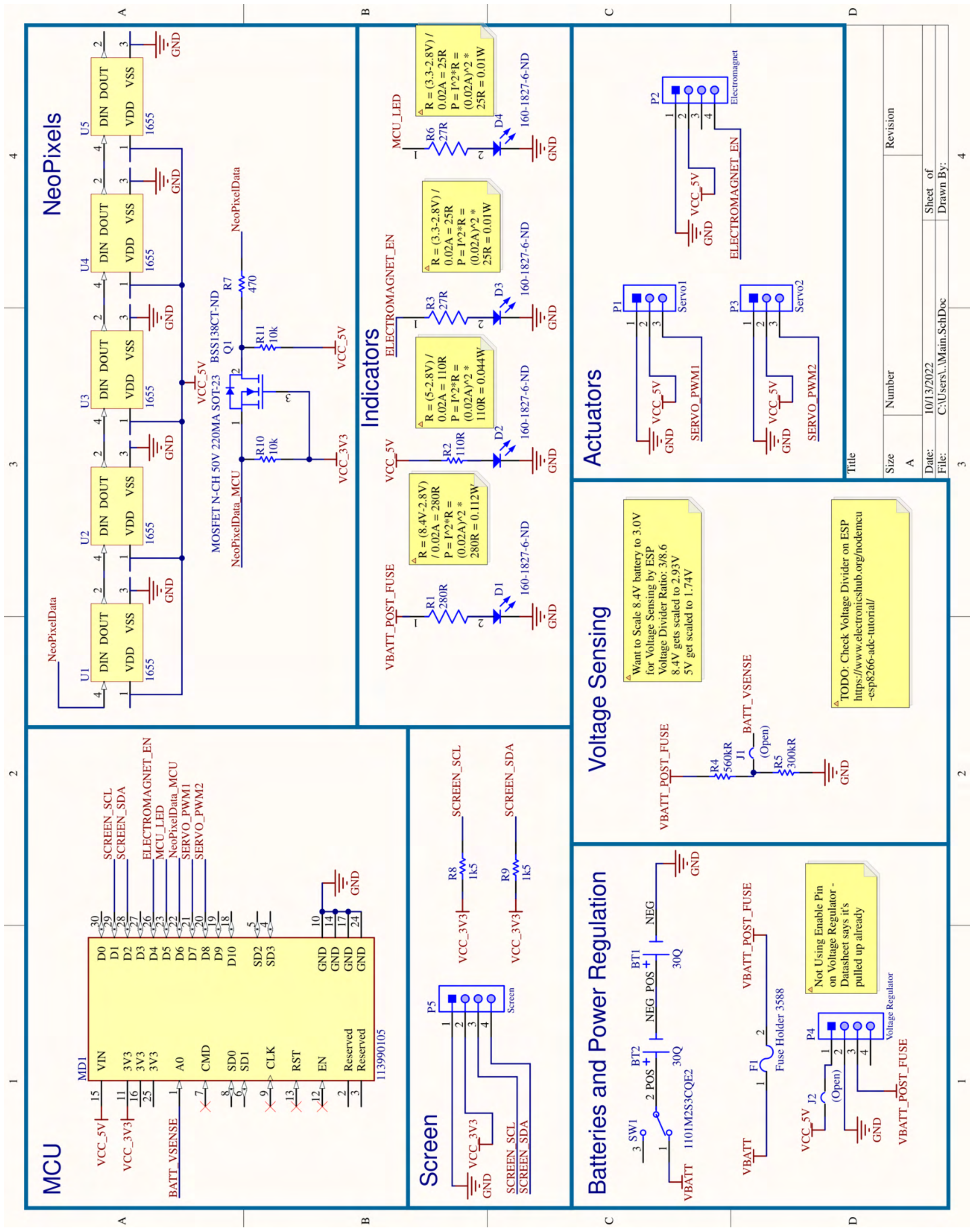

MCU
NeoPixels
Indicators
Screen
Actuators
Voltage Sensing
Batteries and Power Regulation
MD1
11399105
VCC_5V
VCC_3V3
BATT_VSENSE
SCREEN_SCL
SCREEN_SDA
ELECTROMAGNET_EN
MCU_LED
NeoPixelData_MCU
SERVO_PWM1
SERVO_PWM2
GND
NeoPixelData
MOSFET N-CH 50V 220MA SOT-23
BSS138CT-ND
VBATT_POST_FUSE
R = (8.4V-2.8V) / 0.02A = 280R
P = I^2*R = (0.02A)^2 * 280R = 0.112W
R = (5-2.8V) / 0.02A = 110R
P = I^2*R = (0.02A)^2 * 110R = 0.044W
R = (3.3-2.8V) / 0.02A = 25R
P = I^2*R = (0.02A)^2 * 25R = 0.01W
160-1827-6-ND
Want to Scale 8.4V battery to 3.0V for Voltage Sensing by ESP
Voltage Divider Ratio: 3/8.6
8.4V gets scaled to 2.93V
5V get scaled to 1.74V
TODO: Check Voltage Divider on ESP
https://www.electronicshub.org/nodemcu-esp8266-adc-tutorial/
Not Using Enable Pin on Voltage Regulator - Datasheet says it's pulled up already
1101M2S3CQE2
Fuse Holder 3588
VBATT
Electromagnet
Servo1
Servo2
Voltage Regulator
Title
Size
Number
Revision
Date: 10/13/2022
File: C:\Users\..\Main.SchDoc
Sheet of
Drawn By:

Figure 15: Schematic of the Robot PCB.