PORT: Autonomous Port System

Kathy Min, Alan Monge, Jesse Cheung
songheemin@berkeley.edu, alanmonge@berkeley.edu, jessecheung@berkeley.edu

ABSTRACT

Our project, Port, is an autonomous port system that can load and unload cargo to an autonomous vehicle and automated crane system. The goal of our project is to recreate a small scale automated port where autonomous vehicles load and unload cargo at various locations. To achieve this goal, a robust and consistent PID controller needed to be developed for path following and wireless two-way communication needed to be established for multiple devices. Finally, we needed a system of cranes that were able to receive wireless signals from the vehicle to trigger the loading and unloading of cargo.

Video of final system: Crane System, Line Following

1 System Integration Overview

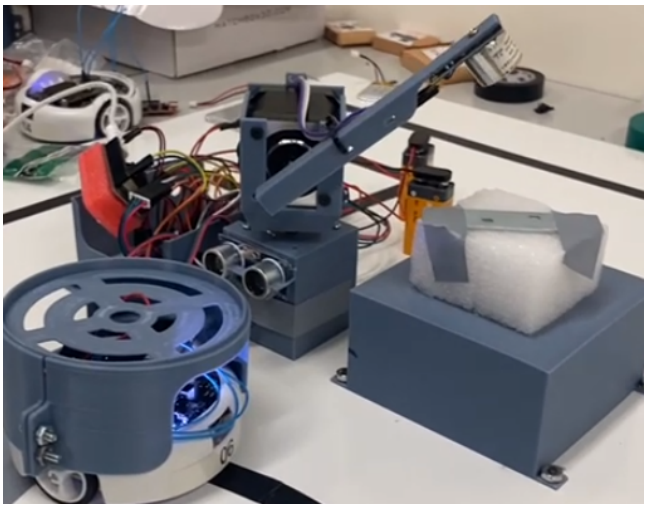


Figure 1: **Picture of the Cargo Vehicle with the Crane**

The major components of our system are the cargo vehicle, cranes, and wireless communication. We integrated the crane and cargo vehicle by having them communicate with each other using letters through ESP NOW, a wireless communication service that is built into ESP32 chips.

The 3pi+ Pololu Turtle robot is used as the autonomous cargo vehicle because of its on-board line sensors which were used to develop a path following algorithm. Since the 3pi+ lacks the ability to send information wirelessly, we chose to have the 3pi+ interface over UART with an ESP32 since it has on-board wifi communication in the form of Bluetooth and ESP-NOW. In our implementation we opted to send wireless signals over ESP-NOW because of its ease of use and ability to send data to multiple devices at once. When communicating between devices, we distinguished between different commands by sending different letters via our wireless communications.

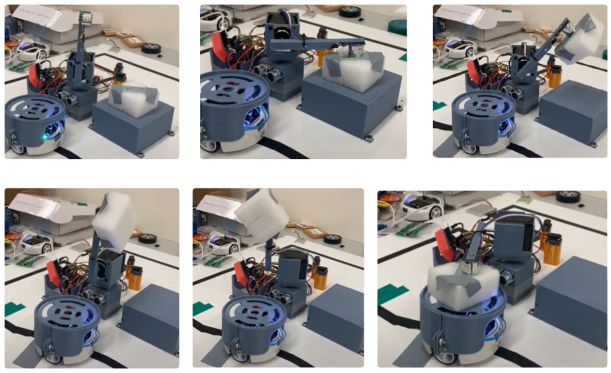


Figure 2: **Visualization of Cargo delivery to the Vehicle**

Our crane system consisted of two cranes, each with two degrees of freedom, an electromagnet attached to the end, and an ultrasonic sensor. The ultrasonic sensor provides another step of verification and ensures that each crane starts operating at the desired time. We opted for stepper motors as our actuators since we needed to precisely return back to the same orientation whenever loading/unloading cargo. Each crane system was controlled by an ESP32 to allow us to easily receive and send wireless signals to and from the vehicle and to easily implement the various sensors and actuators through the GPIO pins. Almost all of the hardware used for our system (not including sensors and actuators) were CAD-ed and 3D printed ourselves.

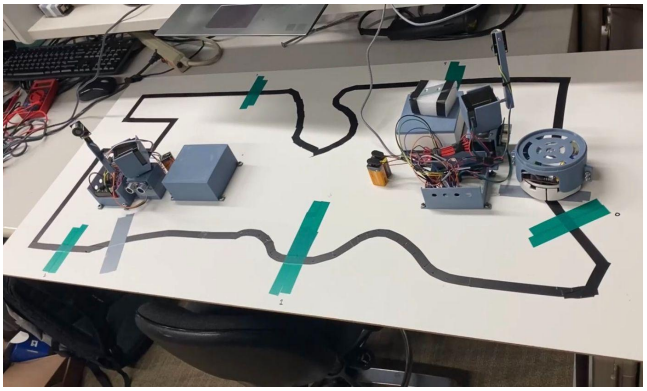


Figure 3: **Picture of the Map**

The map used to represent our port was a flat platform that had a black line for the cargo vehicle to follow. We used gray lines to represent package pick up and drop off zones so that the cargo vehicle could stop for the crane to perform its task. Green lines were used to indicate a new “zone” for localization of cargo vehicles. By identifying which zone the vehicle is in, it would be able to communicate with other cargo vehicles (multi-agent control) and always ensure that the vehicles are at least one zone away from each other.

2 Autonomous Cargo Vehicle

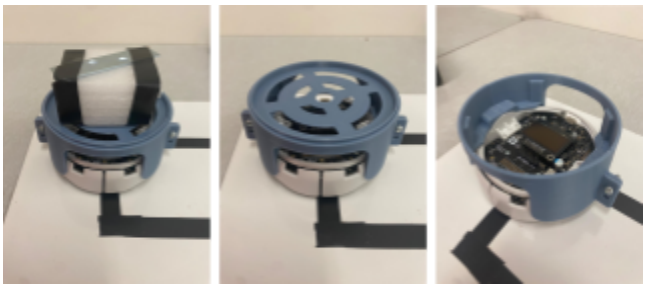


Figure 4: **Cargo Vehicle with Cargo**

2.1 Vehicle Hardware/Software Stack

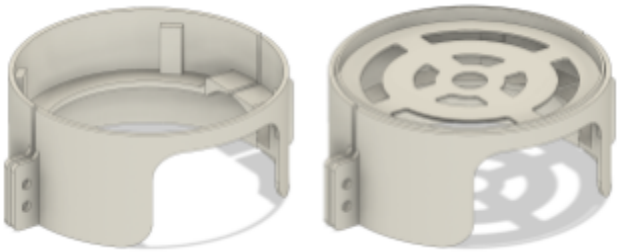


Figure TBD: **3D Model of the Cargo Vehicle**

In order to allow for the Pololu to carry cargo without damaging the vehicle, we designed and 3D printed a perfectly fitting cover with a platform that cargo could be loaded on.

The 3pi+ Pololu runs on a RP2040 microcontroller which is powerful enough to run the needed software for lane control. The software stack that we used are the pico-sdk, Arduino libraries, and Lingua Franca. We used Lingua Franca to handle real time controls regarding timing loops and what modes the vehicle is in. We used Arduino libraries and the pico-sdk to handle the memory-mapped IO interface between the various sensors and actuators.

2.2 Vehicle Timing

Timing impacts how quickly the various sensor values can be updated – most importantly the line sensor, gyro, encoder, and button press values. We used Lingua Franca to handle our timing issues since Lingua Franca gave our software the notion of time. Faster data sampling rates were effective in increasing the smoothness and precision of our vehicle since the increase in reaction triggers allowed for more finite control of our vehicle. However, other factors we had to consider when selecting our optimum timing schemes were not going over the data sampling rate limits of the hardware themselves and ensuring that the speed of computations performed on the vehicle were fast enough for the sampling rate. With the chosen timing values, our car was very effective at line sensing and had very high success rates at round corners and sharp turns.

Our timing values are as follows:
main() - 25ms
gyro - 25ms
encoder - 25ms
line sensors - 25ms
button press - 100ms

2.3 Lane Colors

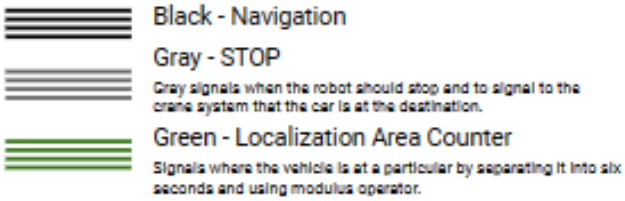
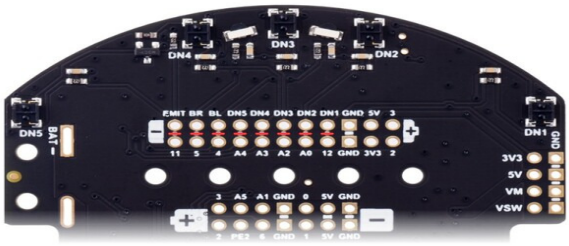


Figure 5: **Overview of lane colors**

The line sensors of the Pololu allowed us to easily create conditionals related to various colors on the map. A gray line was used to indicate when the cargo vehicle should

stop (when it arrives at a package drop off or pick up zone) and for this STOP mode, we required all of the line sensors on the Pololu to read between 400 and 900, the general range of values for the gray color. Green lines were used for detecting entrance into a new zone of the map [zones 1, 2, 3, 4, 5], and the conditional for entry into this mode was for all of the line sensors to read between 150 and 400, the range of values for green. Lastly, the black line was used for general navigation, the DRIVE mode, and this was the default mode if other conditional statements were false. For the DRIVE mode, we used a PID controller with the error being the difference in line sensor values on the right and left of the Pololu. The controller we implemented for the general navigation is expanded upon in the next section.



[1]

Figure 6: **Picture of the 3pi+ Pololu Line Sensors**

2.4 Closed-Loop Feedback System Lane Tracking

The feedback controller that we use for the general line following is a proportional-integral-derivative (PID) controller, which calculates new control outputs using the sum of three terms related to an error we define for the control system. We defined the error to be the difference between line sensors on the right and left of the Pololu.

$e(k) = [L5(k) + L4(k)] - [L1(k) + L2(k)]$, k = timestep

An error value of 0 would mean that the vehicle is perfectly centered over the path line. The implementation of the L5 and L1 sensors was especially effective for cases when the car would veer slightly off course (during sharp turns) and the L2 and L4 sensor values were no longer very meaningful from the car being off course.

The new output was then equal to the sum of three terms: one term proportional to the error, one term related to the integral of the error, and the last related to the derivative of the error. This output was then the desired speed of the left and right motors, which was our input to our MotorsWithFeedback controller provided in the course labs. By implementing the MotorsWithFeedback controller, we could accommodate for the varying weight of the cargo vehicle as it takes on and takes off its packages.

$$p(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}$$

Figure 7: **Proportional Integral Derivative Equation**

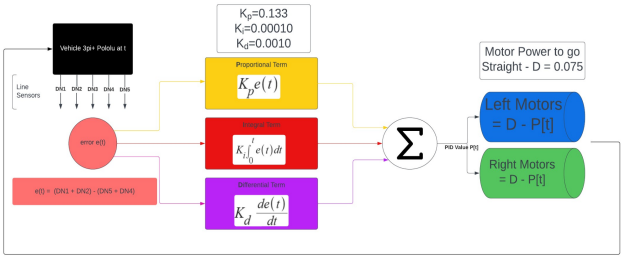


Figure 8: **Overview of the PID Controller**

The Kp, Ki, and Kd constants were tuned by first implementing and optimizing the proportional control only, and then adding the integral and derivative terms to optimize the response. The proportional gain alone helped the stability of the system and allowed the car to follow straight and curvy lines fairly well, but the P-controller was not sufficient enough of a control for the car to reliably make sharp and tight turns. The integral term acted to minimize the cumulative error, which especially helped during sharp turns when we wanted “prior” measurements to impact the control (the car might not be able to make perfect adjustments in time due to the momentum and inertia of the car going straight and might stray from the car otherwise). The derivative term acted as a dampener on our control effort, but we kept the derivative gain to be low since the derivative term was highly sensitive to measurement noise.

With this PID controller, the cargo vehicle was able to handle most disturbances to the path such as turning, small obstacles, and considerable disturbances in the environment (physical pushes).

Other controllers we had originally iterated through were open-loop and bang-bang controllers, but we concluded that the PID controller was the most effective and reliable for all the variations in the map we wanted to implement.

2.5 Vehicle Localization

Figure 9: **Cargo Vehicle on the green line**

$int\ numofareas\ =\ 5$

$position\ =\ location\_counter\ \%\ numofareas$

$\%\ -\ modulus\ operator$

We separated our port into five distinct zones. To understand where its physical location is, the Pololu counts each green line that it passes and updates a counter. By using modular arithmetic, we easily modeled our looped map. Using this localization, we were able to create logic so that multiple cargo vehicles could be on the map at the same time without colliding into each other using wireless communication between the various cargo vehicles. Essentially, since the cargo vehicles know their respective zones, they could send information about their zone to one of the cranes which then compares zones of each of the vehicles. If the difference in zone count between any two vehicles is one or less, the vehicle that is behind should stop until the vehicle in front reaches the next zone.

$while\ ((car1zone\ -\ car2zone\ <=\ 1)\ OR$

$(car1zone\ =\ 0\ AND\ car2zone\ =\ 4)\ ){$

$car2\ STOP$

}

$while\ ((car2zone\ -\ car1zone\ <=\ 1\ OR$

$(car1zone\ =\ 4\ AND\ car2zone\ =\ 0)\ ){$

$car1\ STOP$

}

3 Automated Crane System

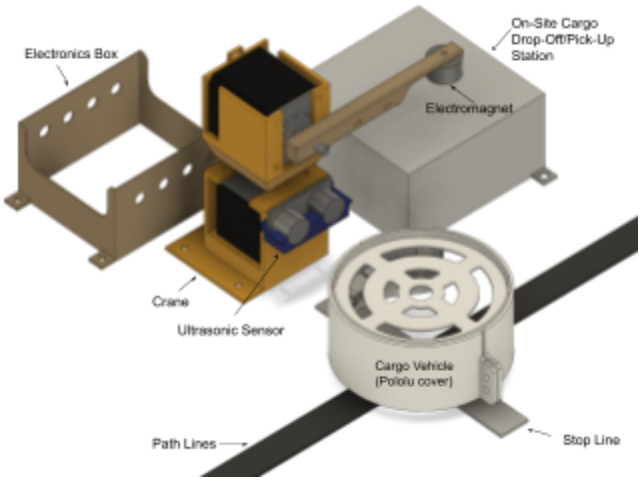


Figure 10: **3D Model of the Crane and Cargo Vehicle System**

There were two cranes in our total assembly: one for dropping off packages on the cargo vehicle and one for picking up packages from the cargo vehicle. Each crane had two degrees of freedom – rotation in x and z – controlled by two stepper motors. With these two degrees of freedom, the crane is able to lower and raise an arm attached to the upper motor (Motor 2) and rotate in z to a new location on the plane the cargo vehicle is driving on.

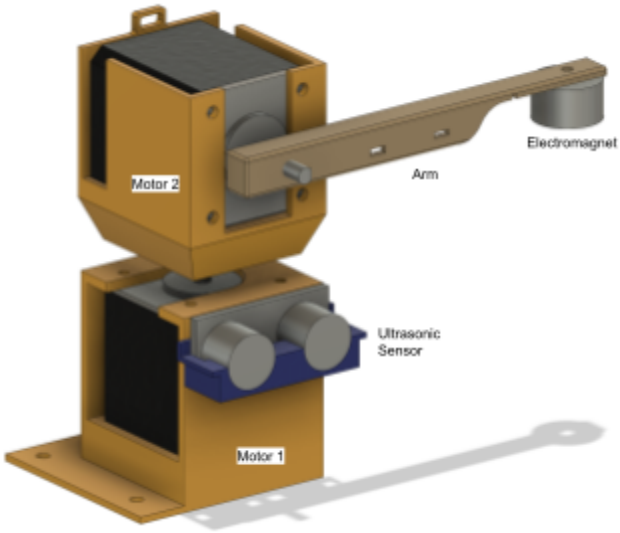


Figure 11: **3D Model of the Crane**

We considered various types of methods for picking up packages, including mechanical grippers and suction cups, but we decided that the most effective method for the purpose of our project would be to use an electromagnet. The electromagnet was implemented with “cargo” that was ferromagnetic (metals are not magnets themselves but are attracted to magnets). When we wanted to pick up the package, we could simply provide a HIGH signal to a logical pin on the electromagnet module to make the electromagnet be magnetic, and when we wanted to put down a package, we could provide a LOW to turn off the magnetism.

An ultrasonic sensor was attached to the bottom motor (Motor 1). This sensor was used for detecting the distance of objects in front of the crane in the direction of the gray stop lines on our map. The ultrasonic sensor was used for redundancy; in order for the crane to begin moving, it requires the following conditional statement to be true: the distance in front of the crane must be less than 10 cm (meaning the crane is in front of the sensor), the cargo vehicle has to send a wireless signal to the crane’s

microcontroller that it has arrived at a stop zone (the gray line), and it must be in the correct zone based on localization of its position on the map.

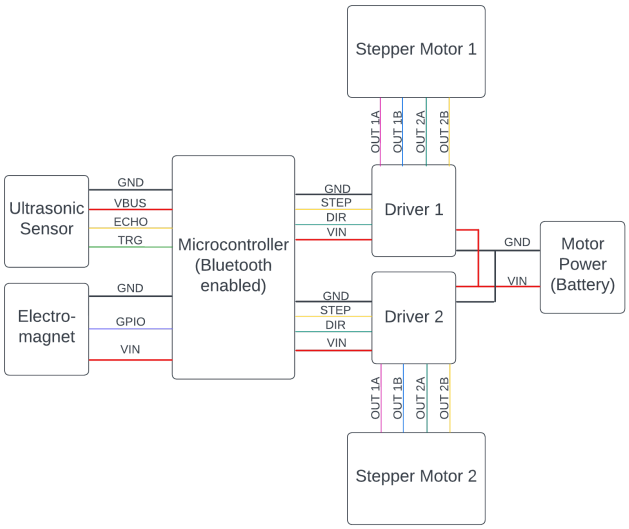


Figure 12: **Diagram of the Crane**

In terms of the electronics, a single ESP32 was used to control the multiple components on each crane. These included the two stepper motors, the ultrasonic sensor, and the electromagnet. Each stepper motor required 9V from a battery so we used a motor driver such that we could supply a high voltage to the motor without damaging components connected to the microcontroller. In addition, the driver we purchased was useful because we could simply output the direction and number of steps we wanted each motor to turn directly to two specific pins on the driver.

A video of our final crane integrated into our autonomous port system is attached to this link.

if (car in STOP zone 1) & (ultrasonic sensor distance <10cm) & (crane1 received wireless signal) {

STOP car

move motors 1 & 2 and turn on electromagnet to pick up package from cargo vehicle

move motors 1 & 2 and turn off electromagnet to drop off package in on-site drop-off zone away from car

enter DRIVE mode

}

if (car in STOP zone 2) & (ultrasonic sensor distance <10cm) & (crane2 received wireless signal) {

STOP car

move motors 1 & 2 and turn on electromagnet to pick up package from on-site package zone

move motors 1 & 2 and turn off electromagnet to drop off package on car

enter DRIVE mode

}

4 Networking/Wireless Communication

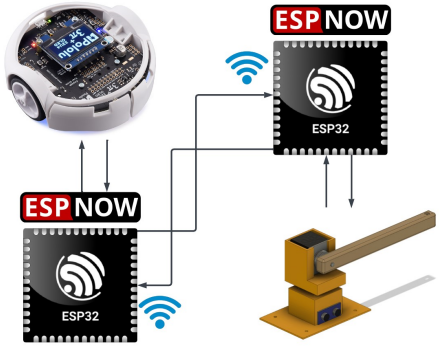


Figure 13: **Diagram of Wireless Communication**

The 3pi+ does not have on-board wireless communication, therefore a wireless enabled device needed to be connected in order to send signals to and from the cranes. We settled on using ESP-NOW to communicate wirelessly with other microcontrollers but originally attempted to implement bluetooth and radio signals. We decided on ESP-NOW because Bluetooth did not allow us to pair with multiple devices easily and radio signals were too sensitive to noise to consistently work. ESP-NOW was simple to implement, only requiring an IRQ for when data was either sent or received, and the mac addresses of any device we wanted to send information to. An ESP32 with ESP-NOW capabilities was connected to the 3pi+ through UART. When sending a signal over ESP-NOW a signal is sent to every registered mac address. Because of this there is no way to send a signal to only one specific device so methods of verification were required to make sure that specific cranes should be activated at the correct time.The 3pi+ sent a different UART signal to the ESP32 everytime the line sensors detected one of the gray or green lines. Unique signals were used to distinguish between gray and green lines. The green lines were used as a localization

method to get a rough idea of where the vehicle was. Every time the vehicle passed over a green line a WiFi signal was sent corresponding to which zone the vehicle entered. Originally, we had planned to have two vehicles following the same path and performing similar tasks, the localization signals sent between ESP32s would ensure that if one of the vehicles was one zone behind the other, it would yield until the other vehicle entered the next zone. When a vehicle detects a gray line, it stops and sends a signal to a crane, telling it to load/unload a package. Because we couldn't write to just one, information from the ultrasonic sensor and the current zone the vehicle is in was sent to each crane as well to verify that a crane should be loading/unloading. Finally, when a crane would finish its task, the ESP32 controlling the crane would send a signal over ESP-NOW that would make the vehicle continue following the path.ESP-NOW was a very reliable method of sending and receiving information, even allowing for simple two-way communication between ESPs.

5 Conclusions

We were able to successfully develop each modular component of our autonomous system, from path following to localization to wireless communications to crane wiring and assembly. We learned very valuable skills through this project, particularly on developing a wireless network system across multiple devices (two cranes and two cargo vehicles cross-communicating with each other), integrating hardware like sensors and actuators with software, and implementing theory learned in class like finite state machines in the real-time computations. Due to the multiple moving parts involved in our project, we found it challenging when integrating all of our modular components into a seamless system, and this is something we would like to work on further in the future.

ACKNOWLEDGMENTS

We thank Victoria Tuck (UC Berkeley) and Pei-Wei Chen (UC Berkeley) for valuable thoughts on the project. We thank Marten Lohstroh (UC Berkeley) for help in answering Lingua Franca related questions. We also thank the UC Berkeley Fall 2023 EECS149/249 course staff and the resources that were made available such as the hardware.