University of California, Los Angeles

EE 183DB Spring 2018

Laundry Folding Robots

Multi-Robot Task Allocation

Jonathan Chang, Michael Moon, and Nick Bruce

Team
Celery

June 15, 2018

Abstract

A project was planned over a 10 week period of using mobile two robots for purposes of laundry folding. While baseline target of making a single fold was met, overall success was mixed per original expectations. An emerging theme of the nature of setbacks were identified. This paper details the experiments conducted.

Background

Although mathematical modeling of mobile robot systems is well-developed, computation regarding complex interactions with deformable surfaces has only recently been made practical due to advances in processing speed. As such, this is an area of active research. One application is automatic laundry folding, usually achieved through large-frame humanoid robots imitating two arms. An early example was BRETT, the Berkeley Robot for the Elimination of Tedious Tasks, by professor Pieter Abbeel's lab in 2003[3]. However, only around 2010 did advances in neural networks and reinforcement learning provide the framework necessarily to accomplish this task in reasonable time. More recently, a commercial project called Laundroid[1] attempted to reduce the size of the arms to fit within a self-sorting closet. Initial release is expected to be priced at $16,000 over the next year.

From a design perspective, the unifying property of these projects are arms capable of full degrees of freedom. Another active area of research, swarm robotics, produce hope of reducing costs. This is using multiple robots that perform simpler tasks that together achieve more complex tasks. Swarm robotics is analogous to the trend of distributed systems over supercomputers, or boosting or bootstrap aggregation over a single, powerful algorithm. This modularity mimics how systems respond to stimuli in nature with decentralized control. Researchers have used this format to model bio-mimicry of bacterial foraging, and terrain coverage and flocking of biological systems[2].

Motivation

Articles of clothing are relatively flat across one dimension, so conceivably the necessary degree of freedom along that axis is short. In other words, full range movement should not be required to achieve the task. Our hypothesis is that laundry folding is an ideal use case for swarm robotics, such that multiple smaller mobile robots could reproduce the required functionality of two full range arms.

The purpose is to reduce cost and complexity; therefore, development life cycle increases in frequency, and the system becomes commercially accessible to a lower end market. Although automatic laundry folding is a luxury good at this time, the task is useful to all market segments. Moreover, research advances in swarm robotics could be applied to the benefit of numerous other tasks.

Problem Statement

Folding laundry isn't just for the wealthy. University students need to fold laundry too. We propose that with a multi-robot system, it is possible to fold laundry automatically, at a cost on the order of $100, as opposed to $1,000, or $16,000.

Identifying Two Types of Problems

The remaining sections of this paper will be written from the perspective after completion. For expediency, we interject the classification of two types of problems we encountered, if we are to be honest: *interpersonal* and *factual*. The

latter is normally discussed here forth; however, while the former is usually ignored to put up the guise of professionalism, it should not be. For the lesson of how to conduct projects as a team is equally important, if not more so, as the lesson of this particular project conducted by this team. Therefore, prominent interpersonal issues will be brought to attention using a gray box.

Initially Identified Scope and Challenges

Initial scope was primarily designed on the basis of cost and an eventually settled understanding that the goal should be to demonstrate a specific, original task, rather than to build a polished end product. Given that project budget was specified to be under $100, our research into prospective prices led us to greatly limit the scope. We noted that success in folding a particular type of garment could be replicated to other types using a similar process. We also deemed the organization and spreading out of garments into initial position to be a related, but different problem. The folding of one type of garment from a predictable, spread-out position marks one important discrete task of an end product.

In order to hit our target budget, we required liberal reuse of parts already owned, and 3D printing or machining of parts we didn't have. Specifically, we had marked the wheels, chassis, and grasping arm, the end effector that interacts with the garment, to be 3D printed.

With our initial understanding of the scope, our predicted challenges revolved around sensing and interaction with the garment, successfully grasping desired layers of it, achieving multiple folds, and detecting grasping points. Other perceived challenges were smooth path trajectory with minimal sensors, although a later question would have been, How minimal?

Sacrifice between constraints and goal caused confusion on how to approach parts of the project. Our initial definition of scope seemed sound only before it was tested against reality.

Disagreement of Scope

Our discussion of scope was contentious due to vague specifications of project focus. There was a lengthy discussion over the metric of what was considered a desirable project. Various ideas proposed juxtaposed systems that merely regurgitated some sensor reading, systems that achieved some industrial task, and systems that interacted with the environment in some intelligent way. Intelligence, or even interaction, was not well-defined in this context, and the perception that other groups' projects fell into some of these categories caused some discontent.

One member of the group proposed using a large metal frame that conducted computer numerical control (CNC) automation to achieve laundry folding. We decided against that option since increased cost and specific skill required in machining parts cause unpredictable dependencies, and that such a system was deemed to be "dumb" and "less research friendly". Even with the swarm system, we failed to machine necessary parts. However, the CNC idea floated around for the duration of the project, partly since we disagreed on interpretation of what was to be expected.

Initial Expectations

On conception, we expected that we would have been able to complete multiple folds on a standard garment (e.g. a t-shirt) using dual robots. We had expected that sensing and algorithms would be the sticking points. We also noted that a working system would be generalizable due to existing algorithms for better state detection. We expected that such a system would be possible under 100% over budget; that is, within $200. Our market research wasn't broad enough.

Initial Project Plan

Which Plan?

The presentation version of our final plan lacked sufficient detail, so the report had to fill in some blanks. Unfortunately, as it often was the case, conflicting schedules prevented us from agreeing upon a time line to work together. The lack of communication bred mistrust on the amount of contribution. As such, the report was submitted before a member suggested his version, and two versions of the project plan existed. In the first week, group members had disparate understandings of what the plan was.

The project plan submitted in the initial report was not agreed-upon by all members of the group, so a revised version was later submitted, and subsequently followed.

Week 1:

Task	Deliverable	Score	
		Allocated	Achieved
Draw block diagram of components.	Block diagram.	2/10	
Order components.	Receipt.	3/10	
Wiring schematic.	Schematic.	3/10	
Complete machine shop orientation	Signed paper	2/10	
Total		10/10	/10

Week 2:

Task	Deliverable	Score	
		Allocated	Achieved
Design car in CAD (1st iteration)	CAD layout.	4/10	
Receive and check BOM.	Checklist of parts.	1/10	
Design car in CAD (2nd iteration)	Final schematic	4/10	
Begin machining parts	Finished car	1/10	
Total		10/10	/10

Week 3:

Task	Deliverable	Score	
		Allocated	Achieved
Calibrate sensors	Calibrated sensors	2/10	
Soldering	Soldered perf board	2/10	
Machine car parts	Refine unwanted parts	2/10	
Wire car together	Built robot.	4/10	
Total		10/10	/10

Week 4:

Task	Deliverable	Score	
		Allocated	Achieved
Wire up car	Functioning car	4/10	
Calibrate sensors	Std. dev. before/after sensors	1/10	
Covariance matrix	Covariance matrix	1/10	
Color sensing	Measured values	4/10	
Total		10/10	/10

Week 5:

Task	Deliverable	Score	
		Allocated	Achieved
Create robot network	Python code	3/10	
LEDs to check proper communication	Demonstration	2/10	
Setup course for line-following	Demonstration	2/10	
Move synchronously	Demonstration	3/10	
Total		10/10	/10

Week 6:

Task	Deliverable	Score	
		Allocated	Achieved
Develop protocol for robot coordination	Demonstration	10/10	
		/10	
		/10	
		/10	
Total		10/10	/10

Week 7:

Task	Deliverable	Score	
		Allocated	Achieved
Sense relative fabric position	Demonstration.	10/10	
		/10	
		/10	
		/10	
Total		10/10	/10

Week 8:

Task	Deliverable	Score	
		Allocated	Achieved
Pick up fabric	Demonstration	10/10	
		/10	
		/10	
		/10	
Total		10/10	/10

Week 9:

Task	Deliverable	Score	
		Allocated	Achieved
Implement first fold	Demonstration.	10/10	
		/10	
		/10	
		/10	
Total		10/10	/10

Week 10:

Task	Deliverable	Score	
		Allocated	Achieved
Implement second fold	Demonstration.	10/10	
		/10	
		/10	
		/10	
Total		10/10	/10

Finals Week:

Task	Deliverable	Score	
		Allocated	Achieved
Lab report	Lab report.	10/10	
		/10	
		/10	
		/10	
Total		10/10	/10

Sensor Considerations

In order navigate around garment, some kind of state estimation is required. Designs with an external camera system would have been ideal, but was rejected early on cost. Several more economical options exist.

Encoders are a reliable method to track state. There are primarily two types of encoders: Hall and optical. The prior uses Hall Effect to detect a flip in magnetic poles when the wheel turns. Its resolution, then, is limited to a turn of the wheel; however, this is ameliorated by smaller to large gearing, such that the small gear turns multiple times every turn of the wheel. This has implications on the maximum speed and load on

the motors.

Optical sensors use a black and white pinwheel and a light sensor to detect changes in color. Its resolution is dependent on the accuracy of the sensor and noise rejection algorithm, which would allow for more or less stripes on the pinwheel. Resolution down to 1 degree is possible.

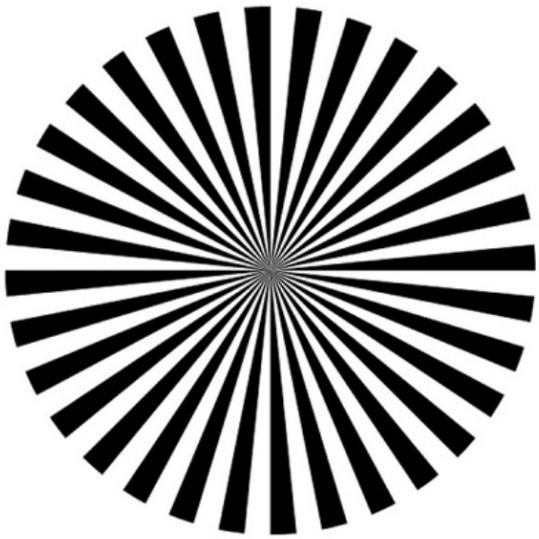

Figure 1: An example of an optical encoder wheel.

9-DOF inertial measurement units (IMU) incorporate an accelerometer, magnetometer, and gyroscope on a single chip, usually for state measurement. They are of varying cost depending on precision, but we already had one. The magnetometer and gyroscope achieves similar function: they track the absolute and relative to start angles respectively. The former may act as a compass in absence of interference. While the performance of the magnetometer is unpredictable, the gyroscope is known to drift over time. The two could be fused using an algorithm such as the Kalman filter to reduce error. The accelerometer, in this instance, tracks xy velocity and position by integrating over acceleration. z-axis could use gravity to track field level, but we presume a flat and level surface. State fusion using Kalman greatly increases algorithmic complexity, which introduces inertia of another kind – inertia of human action.

A more rudimentary method uses color sensing to track contrasts of light in the environment. Presuming that the field is a single, predictable color, garment must be a color of sufficient distance. Robots must navigate near the edges of the garment if only color sensing is used without losing track of state. However, this comes with some hefty assumption. For one, to ensure reliable operation, robots must run on certain surfaces that precludes hard wood or some carpets. A constructed field might be necessary for broad operation, which partially counters some of the design goals of this project. Of course, multiple sensors should be incorporated to provide better or redundant state estimation.

Our color sensor *Adafruit TCS34725* came in 4 filters: red, blue, green, and clear. The filters are presumably implemented by sensing refraction from a bright white LED light situated on the chip. Analog values are 0-65535 ($2^16 - 1$). Standard library by Adafruit includes functions to read color temperature (K), lux, and the brightness seen by the 4 filters. Red light is a bit harder to detect, which is expected. White materials consistently show near maximum values on blue, green, and clear filters, but less on red. Black plastic materials do not absorb light completely; therefore, the sensor shows about 3000 on the clear filter. Other colors give intermediate values.

The exact color combination of the RGB filters are unreliable, probably due to biases in ambient lighting and other nearby materials, such as flooring or walls. However, different colors can be determined by comparing relative ratios of RGB, as well as overall brightness. This requires posterior experimentation. Preliminary tested materials include various white napkins and papers, black electrical tape, mouse, and calculator, and a piece of green fabric.

Results are dumped into the Arduino serial monitor, and copied into Excel for further analysis.

	Color Temp (K)	Lux	Red	Green	Blue	Clear	Material	Distance
2	65535	0	0	0	0	0	INIT	N/A
3	1681	106	484	253	185	871	X	N/A
4	1681	106	484	253	185	871		
5	7388	45443	49275	65535	57380	65535	Napkin	Up close
6	7246	45151	48477	64559	56028	65535		
7	4359	571	1215	1005	847	3093	Electrical Tape	Up close
8	4370	576	1223	1013	854	3115		
9	5300	1056	2217	1924	1722	5926	Fabric	A few inches
10	5227	1049	2179	1894	1684	5832		
11	5304	3066	5215	4979	4234	15350	Fabric	An inch
12	5145	3318	5738	5396	4557	16778		
13	4080	6480	13743	11200	9202	36047	Fabric	Up close
14	3814	6407	14522	11359	9299	37011		

Figure 2: A snippet of color sensor values.

A final method even more rudimentary is to use line following along black tape outlined around the garment, which makes ability to generalize infeasible, as special consideration must be taken for each piece of garment with respects to size and position. Since this method greatly limits scope, it may also greatly limit design complexity. Our eventual experience dictates that there are other bottlenecks regardless of this decision; however, the decision in the first place underscores potential misalignment with initial expectations.

Drive Train Considerations

Several design choices were made with respect to the drive train, but factors into the final decision were dwarfed by cost. From basis of prior projects, using 2 wheels seems like an obvious choice. It has advantages of being inexpensive, contributing less friction compared to 4 wheels, due to the differential, or lack thereof. Primary disadvantage is the stability, probably desirable due to the grasping assembly off to one side.

A ball caster in the front center could contribute to stability. For 2 wheels, this is highly desirable at low cost.

4 wheels is another possibility that has high controllability, makes possible more design configurations. The primary advantage is that wheels could be further apart and still maintain stability. However, due to lack of differential, wheels would introduce a small amount of friction when turning. It also adds to expense.

Using either 3 or 5 wheels, we could use omniwheels to allow holonomic movement, such that it could move sideways without turning. This helps to decrease grasping complexity and shift the functionality of extension into the drive train. However, omniwheels are incredibly expensive, and most wheels on the market would require additional hubs and more powerful motors. There's a cascading Domino's effect on cost, which will be discussed later.

Considerations on the Arm Design

The grasping arm was initially noted to be an area of difficulty. Due to the unpredictability of fabric stiffness and folds, robots must employ several measures to maximize generality. Ideally, there should be at least the following 3 degrees of freedom.

(a) The arm should *extend* to scoop under the fabric and allow some delta between robot and fabric edges. This adds robustness, since exact robot positioning should not be expected. Wheels should not tread on the fabric; however, increased distance contributes to imbalance, and requires greater counterbalance weight.

(b) The arm should *lift* the fabric off the ground to encourage folding as opposed to pushing against itself. Greater lift should counteract stiffness. Project specification required limited z-axis movement, yet lift relative to robot height is necessary.

(c) The arm should *turn* the fabric over so as to constitute a fold. Lighter fabrics may fold without a turn. Heavier fabrics may require one given limited lift.

The arm should also have the ability to measure tautness of fabric so as to ensure robots move at an appropriate distance. Using a line-following design removed this necessity, however at an obvious cost to generality.

Without additional constraints, such a design may be conceived with relative ease. However, factoring in cost, size, weight, and power

constraints greatly complicates design. A laissez faire design might require 3 motors, one for each degree of freedom, and a pressure sensor. The arm might use some combination of plastic and metal that works around the motors. And the microcontroller should, of course, deliver requisite power. However, all these areas are limited. Our initial $100 budget assumes only 2 motors for this purpose and entirely 3D printed parts. Complexity correlates with mass, and the imbalance requires the chassis to be upgraded to handle the additional weight and size. Moreover, given just one battery, the ESP-8266 motor shield provides limited slots for independent motor control.

Compromise comes in several forms.

(a) *Underactuation* uses gearing and ingenuity to achieve goals with less actuation than degrees of freedom. Much consideration for this was given over a couple weeks; however, the cost of gears and pulleys to make this possible would have been prohibitive as well. These parts are typically geared towards hobby, competition, although smaller, more advanced designs are found in industrial applications.

(b) *Reducing functionality* may be necessary to hit target budget. Since complexity toward size, weight, and power also contributes to cost, sacrifice to design may be necessary in context of market support. For prototyping under budget, market of scale works against us.

(c) *Re-specification* might have been the most realistic choice. Class structure made this choice more risky, so it was not really considered.

Despite this, we were able to imagine several designs, although sourcing parts to build them proved much difficulty.

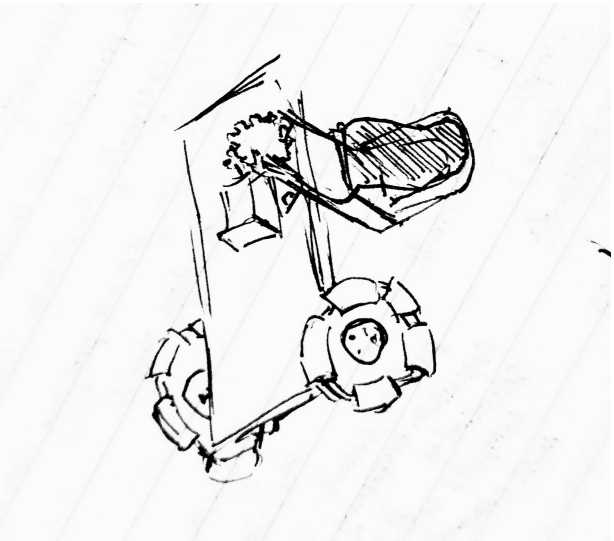

Figure 3: Sketch of one such design.

The State of State Estimation

The scope was originally declared to focus on a specific task while being open to generality. State estimation of some form would have been paramount. However, either an executive decision was implicitly made to disregard generality, or lack of planning made it so. As such, aforementioned considerations seemed tangential at best, in retrospect. For example, mobile robots constrained to drawn lines precludes generality of garments; therefore, most sensor considerations become superfluous. If robots and garments have predefined positions, it would have been redundant to detect grasping points. Similarly, freedom of omniwheels become unnecessary given robots only need drive in a straight line.
Nevertheless, these were considered, even if their contributions were abstract and invisible.

Chassis Considerations

The chassis has only several requisite properties.

(a) It must be an appropriate size to house electronics, sensors, and mount the drive train.

(b) It must be sturdy and inflexible.

(c) It should not contribute much to cost.

The last property relegated the task to 3D printing, although in retrospect, some materials (e.g. drywall) might have satisfied conditions inexpensively and saved time. Creativity might have found another solution besides common sense, but hindsight was 20/20.

We all learned CAD, to some degree or another. While a rectangular box with some extrusions wasn't particularly difficult, we found that printing had tolerances of error that made size imprecise. It took several iterations to become acquainted with discrepancies between dimensions reported by various software and the observable result. The 3D printer used was a public resource, at Maker Space and IEEE, and this compounded with the long print times that made these iterations a lengthy process. Small mistakes caused long delays. The downside to 3D printing is their inability to print fine details coupled with slow speeds and a size limit. In other words, projects could neither be too small, or too large. One member exerted Herculean effort for his perseverance and insistence that allowed our chassis to be printed.

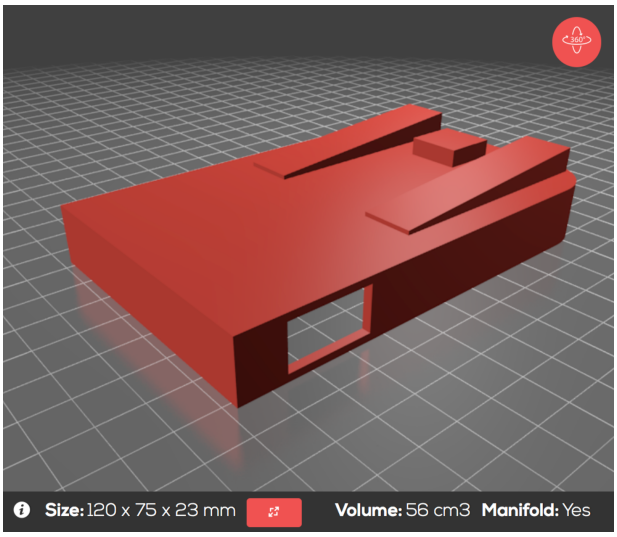

Figure 4: Single iteration of 3D printing.

Wheel Considerations

Our wheel selection went through several iterations as well. Initial designs were 3D printed to minimize cost. However, slight imperfections to the printing caused errors difficult to detect. For example, one side would slip or get stuck because of slight differences in size or irregularities in the surface.

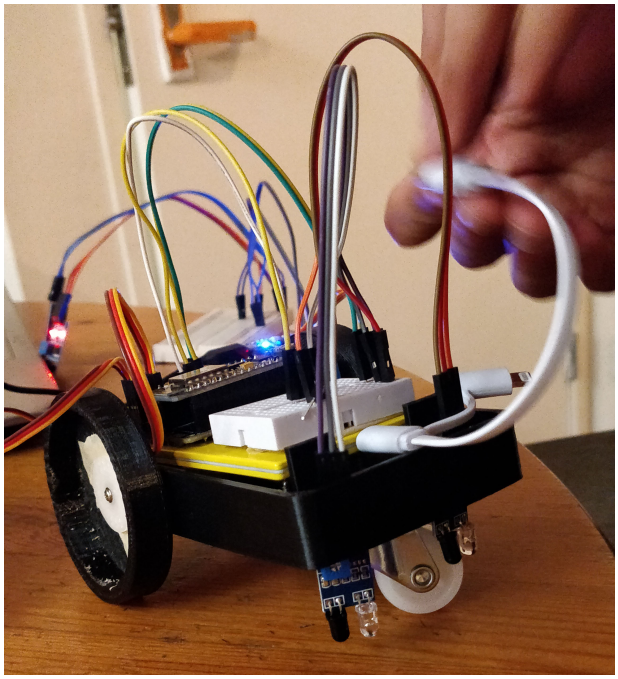

Figure 5: 3D printed wheel.

Subsequent wheel choices relied on finding common household items, which were round, such as bottle caps or tape rolls. This would leverage pre-fabrication to minimize cost. There were difficulties in mounting to motor axes, that, again, caused friction.

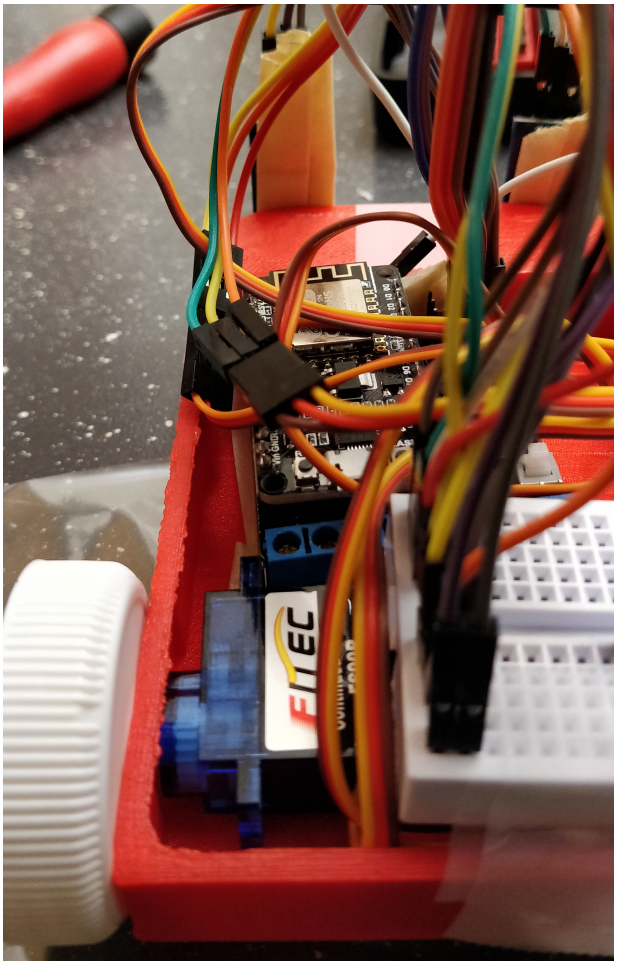

Figure 6: Bottle cap wheel.

Our experience with the difficulties in getting 3D necessary dimensions led us to replace critical and moving parts with pre-fabricated components. This greatly simplifies development, since imperfect wheels cause silent errors that complicates troubleshooting. For instance, stuck wheels look like sensor error, and is usually unsuspected. Therefore, while large uncritical parts like the chassis could be 3D printed to maintain cost, smaller moving parts like wheels and motor assembly should use commercial parts to maintain functionality. To this end, we found inexpensive 60mm wheels and 3/4" ball casters from Pololu. The ball casters replaced pre-owned swivel casters that were too large compared to the rest of the body.

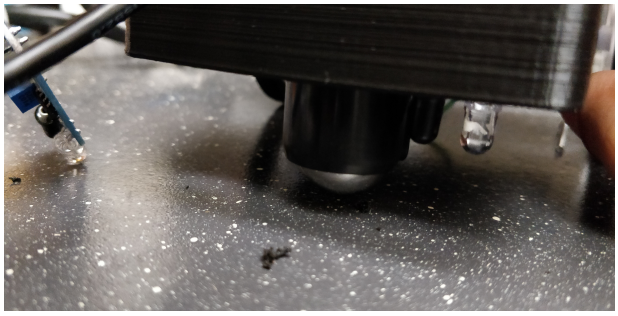

Figure 7: Ball caster replaced a swivel caster.

Further Considerations on Line Following and Sensor Constellations

We do not propose that astrology causes a magnetic field that affects this project. Instead, relative positioning of line following sensors could be thought of as a "constellation" that affects performance. Examination is mostly a priori thought experiments, since the mechanism is simple enough. In this section, we will look at various linear arrangements of 2-4 sensors. Constellations are not necessarily linear, but the robot's ability to sense the line is analogous to grasp closure; therefore, realistically, non-linear formations require at least 4 sensors.

In the following section, we will consider the terminology *wide* when the left and right sensors are far away from the line, so as to maximize sense of rotation; and *narrow* when the respective sensors are close to the line, so as to minimize error. Error is considered the distance between robot center of axis and the line. The exact measurement is not important, as long as we recognize the reason behind placement.

We hold several assumptions. The first is that smooth operation is desirable, such that movement should not be jerky. There should not be any sudden stops. The robot, therefore, should move continuously until ends or sharp turns.

We also assume the line following sensing is in isolation from other sensors, such that orientation is achieved only from this subsystem. This is typically true, since internal state is unaware

of the track, unless a specialized sensor oversees the entire track.

Detailed examination here is necessary because overzealous designers might think 2 sensors are enough. At first glance, this is sensible. However, we suggest that increasing number of sensors provide layers of redundancy preventing the robot from going off course, thereby allowing an increase in speed.

We begin with a constellation we will call *2-wide*, with left and right sensors far apart. This has an obvious advantage over 1 sensor, since the sensors straddle the line, which ensures relatively correct orientation in normal conditions. A single sensor placed in the center would detect (*black*) if the robot is centered. It loses orientation the moment it detects (*white*), since it has no sense of direction relative to the line. It cannot know whether to turn left or right to reorient. Therefore, robots with 1 sensor need to constantly sweep both directions. *2-wide* constellation solves this problem, but another one is immediately introduced.

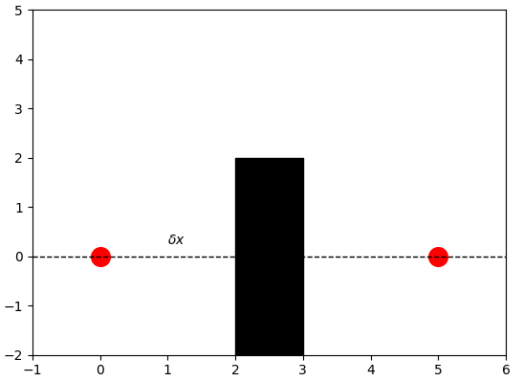

Figure 8: Robot centered on line with 2-wide.

The wideness allows the robot to be sensitive to turns, since 2 sensors cover longer arc length further apart, but it allows for higher error. As shown in *Figure 1* above, there's a relatively large distance the robot has to drift for it to detect that it is off-center. It can be nearly half-width off center and still see $(white, white)$. After seeing $(black, white)$, it corrects itself and is prone to zig-zag after seeing $(white, white)$ again, since it cannot get a sense of the line without touching it on both sides.

Therefore, for smooth response, *2-wide* is untenable.

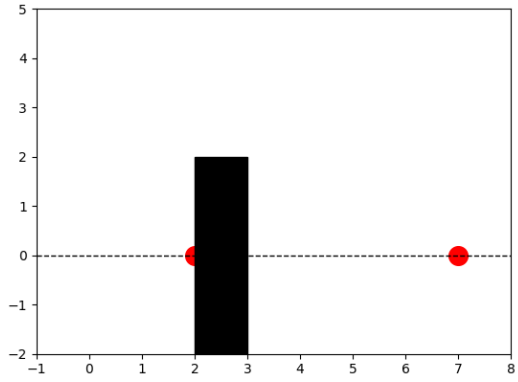

Figure 9: Robot sees black for the first time after drifting half a width.

Next, we will consider a *2-narrow* constellation, with the sensors just outside the width of the line. Clearly, this is meant to solve the problem by minimizing error allowance. However, as we will see shortly, this increases chance of becoming "lost". While *2-wide* has to drift half-width before it self-corrects, it must exceed that distance to lose sense. That is, if the robot sees $(white, white)$, with both sensors on one side of the line, then it loses sense of the line (and thinks it is still going in the right direction). *2-narrow*, however, is closer to this possibility.

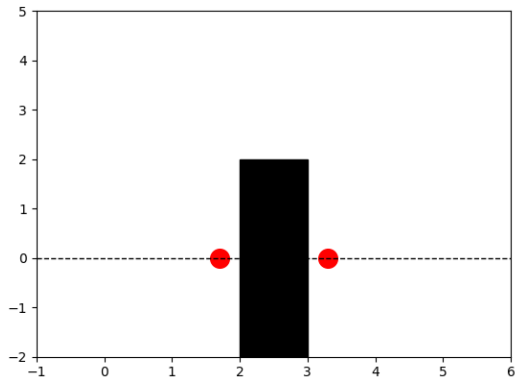

Figure 10: Less error, but less safety.

Problems occur when the robot reaches a sharp turn. Note that despite being in two different situation, the following has the same state.

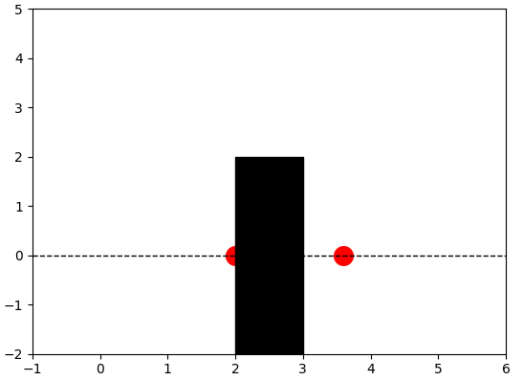

Figure 11: Robot that sees black, white, needs to readjust.

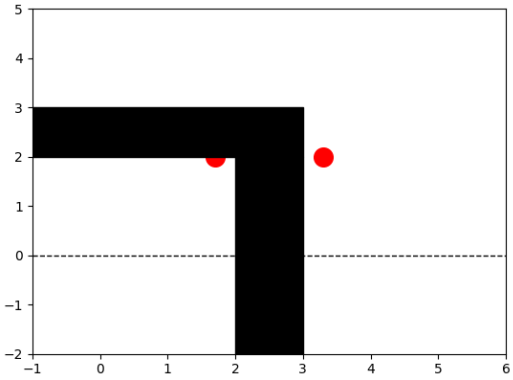

Figure 12: Robot sees black, white, but is in danger.

Given normal operations of smoothly correcting itself by turning left, the robot will lose track of state if speed is sufficient.

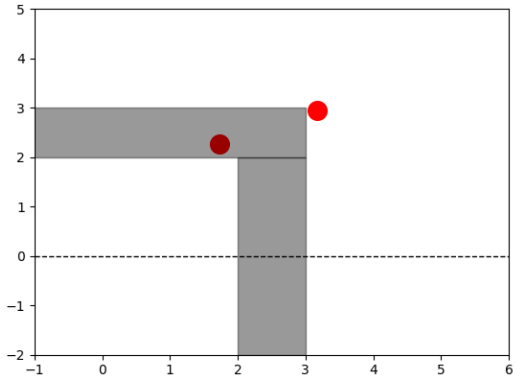

Figure 13: Robot veers slightly left after left sensors senses black.

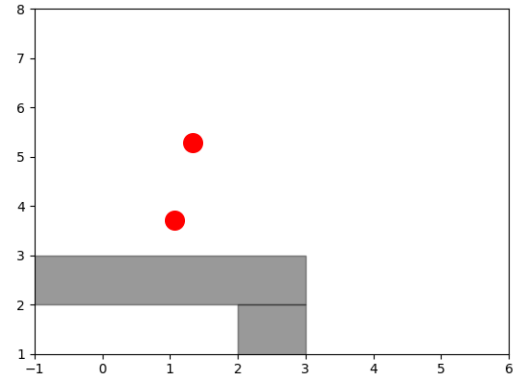

Figure 14: Despite veering left, robot is completely off track.

With 3 sensors, *3-wide* has a different, trivial problem. The robot sees the same state whether it centers on the left or right side of the track.

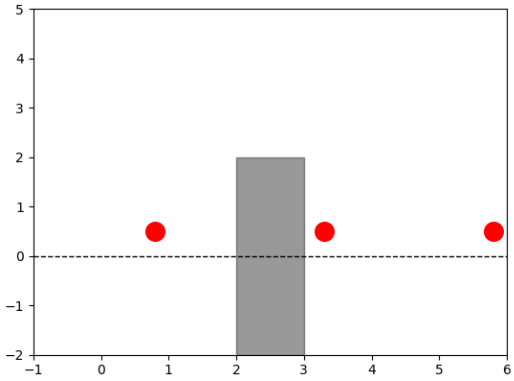

Figure 15: Robot is on the left.

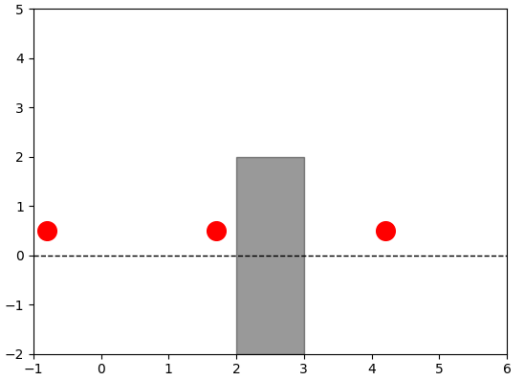

Figure 16: Robot is on the right.

In our thought experiment, a minimum of 4 sensors is necessary to approach a decent speed. However, this assumes the presence of sharp corners. Folding a t-shirt would have encountered such a problem. Folding a scarf or pocket square might not. Therefore, its inclusion goes back to how we interpret our scope.

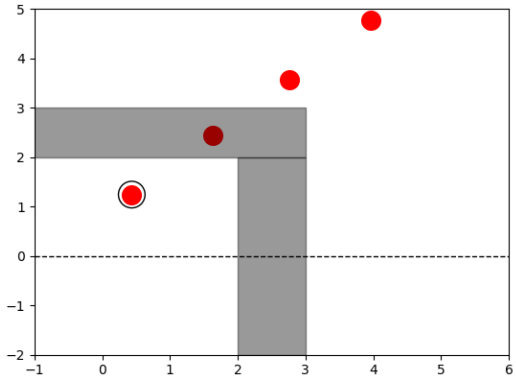

Figure 17: Robot sees black, white, but is in danger.

Considerations After Experience of 3D Printing

In order to realize a design of our own, we considered multiple avenues before settling on 3D printing. In the end, we decided to use the 3D printing option because it was (in theory) faster and more reliable than building with wood and metal via the Engineering Machine Shop in Boelter Hall. At first, we used the 3D printer in the IEEE lab on-campus, and quickly realized the difficulty in troubleshooting the device that we would spend the next 2-3 weeks bonding with. Shown below is the minimal interface that the New Matter Mod-T Printing Utility offered. Upon printing our first iteration of the chassis, this printer stopped half-way through our second iteration and left is without a 3D printer. The company itself, New Matter, recently filed for closure and as such, offered little to no explanation on how to troubleshoot the printer. In an attempt to fix the printer, we ended up reloading filament, plugging it in via USB (rather than connecting via Wi-Fi), and scouring the internet for support forums. After having spent a rigorous 6-7 hour stint on what never amounted to anything, we began to look for other resources.

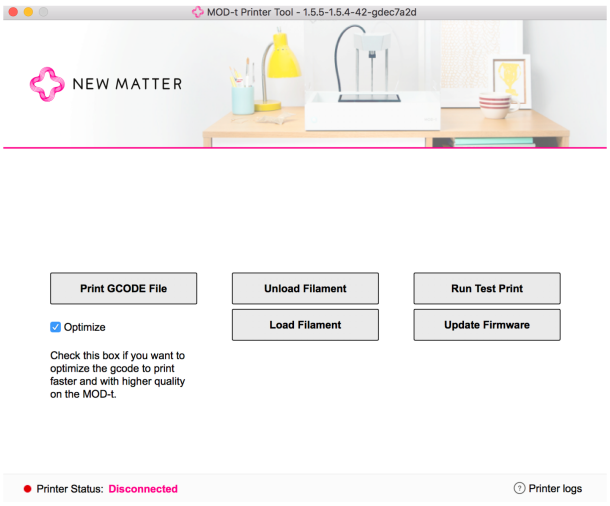

Figure 18: Mod-T printer utility interface lacks troubleshooting options.

First on our horizon was the Maker Space workshop near Rieber Hall. At first, this place offered immediate printing with little to no delay and good employees who were willing and able to troubleshoot or help with any problems we had in the scaling process. After we finished printing our second iteration, we realized that we needed yet another chassis of different dimensions. This time, we returned to the Maker Space to find out that there had been instituted a new queue policy, which prevented any print jobs lasting longer than 3-4 hours.

Unfortunately, this mean that we had to decrease our infill, which is essentially the density, of our chassis to the lowest possible amount: %20 infill. At first, we didnt mind scaling our density down to %20 but would quickly realize that this was too brittle for our car to work reliably and eventually led to instability in the chassis.

Next we sought out the only other on-campus resource which was Lux Lab at Powell. Here, we encountered the easiest, most streamlined way to print 3D parts, which we ended up using for the remainder of the project. For anyone considering 3D printing on campus, Lux Lab is by far the most professional and easy to use venue for 3D parts.

Different iterations of the chassis over the course of our 10 week period are shown below.

The Usefulness of 3D

Although the limitations of 3D printing was apparent rather early, the group was not in agreement over its continued usage. It is difficult to let go of a design after so much investment in it, especially with no discrete alternative. Continued investment, however, locks in one possibility. Perhaps, if everyone was fully invested, we might have done better still. This underscores the importance of group cohesion, perhaps more so than devotion to any one technology as opposed to exploration.

Considerations on Software, Particularly Web Sockets

The ESP8266 includes WiFi functionality; however, the built-in library is limited to hosting a web server and sending data through hypertext transfer protocol (HTTP). While this makes it easier to represent graphical interfaces for end-user interaction, it is not the ideal protocol for server to server communication. Transfer control protocol (TCP) is faster and more direct, with less overhead in decoding packet headers and such. The Arduino WiFi Shield has such a library for expedient implementation. It is incompatible with the ESP8266; however, a more recent Sparkfun shield and software port would have allowed this.

On the other hand, it would be cost-wise preferable to use the internal chip functionality if it's good enough, despite the inconvenience.

We found third-party libraries that enabled TCP through Web Sockets API compatible with the ESP-8266. Sockets are rather low-level objects hidden in modern libraries. Implementation was slightly more complicated. It interfered with the ESP-8266 access point, a known problem.

Design-Cost Trade Off

Original project target was to spend less than $100 in order to deliver a low-cost design, but also to maintain budget. As we progressed, design-cost trade off became readily apparent; in fact, one of the main concerns. Since none of us have spare cash lying around, we tried hard to adhere to this goal, but we no longer expected it to be possible. Functionality would be sacrificed in pursuit of frugality. One obvious area is the grasping arm. As previously discussed, the arm has up to 3-4 degrees of freedom: extension, grasping, lifting, and turning. Implementing these with a separate motor each requires space. Furthermore, picking up articles of clothing that are heavier require more powerful motors and parts to maintain tension. Larger space requirements due to more or larger motors increase size of the chassis, and potentially wheels if more space is required underneath. Since this comes at increased weight, more powerful motors are necessary to power the drive train. In turn, we might require a more flexible controller than the ESP-8266. This cascade of changes turns a $100 project to a $400 project easily. We cannot just over-design one aspect without influencing all others.

On the other hand, reducing number of motors cause either reduction in functionality, or increase in design complexity. And so we've spent several weeks working around a general lack of available parts and budget, trying to invent some complex gearing mechanism with ingenuity to hit all of targets with one stone. We came up with designs of under-actuation, requiring specialized gearing and parts that would be too fine to 3D print, yet prefabricated parts could not be found. We scoured Home Depot for an entire day, and found parts only vaguely resembling the design.

Other relevant areas include the design of the chassis, and inclusion of various sensors. 3D printing both makes possible several design choices while eliminating others, due to a limitation in fineness and size, as well as the complexity of using CAD software, of which we are all novices at the outset of this project.

Design-cost trade off has, therefore, become a major issue, especially with respect to the grasping assembly. The clamp needs a lower jaw thin enough to slide under fabric, and the gearing assembly needs to somehow fit on the side or under the chassis. With little to no experience in mechanical assemblage, our team has been a bit overwhelmed. Over time, we have been forced to pick up other parts, an overall tally of which most likely would exceed budget. In hindsight, it was impossible for inexperienced students to accurately estimate cost anyways.

Meta-progress

Deliverables are some real, tangible, measurement of progress, used in this class and in industry. The difference might be that projects in industry are usually well specified and researched by teams of people with domain knowledge. Industry is bound to deliverables, however, due to a lack of visible accountability in a large-scale multi-level structure. This class lacks that structure, but still lacks the visibility. I argue that due specifically to this lack of structure, significant amounts of progress can be made that is not reflected in deliverables. That is, advances in understanding the problem, and problems in answering the problem are made without adhering to the deliverables defined up-front. Notably, the number of flawed attempts that did not make it in the final product, that were deemed unsuitable to even demonstrate, are invisible, as well as the deliberation in contending with constraints that had to be set before any in-depth exploration. This unrealistic expectation is similar to expecting R&D to build a working experimental prototype on the alpha iteration, and on limited budget.

To be sure, there are two roadblocks here that are relevant everywhere, but especially to students:

(a) Time. Students are necessarily responsible, as dictated by department regulation, for 3-4 classes, whereas industry professionals are paid to specifically do one job. Small team format forces students to

put on all hats, explore all roles, all the while being subjected to expectedly unpredictable external constraints.

(b) Budget. Students do not have corporate budget to pour into research. Specifically, numerous false starts cannot fit into the paltry $100 given budget, and invisible costs such as expedited shipping and trips to hobby stores add up.

This is especially relevant to our fabric grasping arm, which it the quintessential functionality of our specification. The fineness of motor gears could not have been 3D printed. Pre-built are either industrial or marketed toward well-funded education programs and competitions, since hobbyists rarely work with such specific parts. It would have been possible to purchase generic rack and pinion gears. To tailor it to our specifications would have required, perhaps, the entire gear set, if that would even be enough, since prebuilt parts are not exact matches to any individual design. We do not have the CAD expertise to import existing parts and simulate a functional gearing system like mechanical engineering grad students might have.

The lack of time to deep dive down a rabbit hole of unknown depth, and the unwillingness to spend hundreds of dollars meant that we had to inefficiently conduct short, breadth-first searches, and upon realizing impossible constraints, extend the branches and reiterate, until realizing the true meaning of design-cost trade off. None of this is reflected in our deliverables. Perhaps most of it is unavoidable. It would not have been unavoidable if we were not students. For example, we might have paid a foreign factory to build small metal or plastic pieces to our specification.

The inability to rework specifications or clarify goals, whether due to the 10 week time constraint or class structure is, in my opinion, a large handicap. Failure to meet early deliverables cascade, leaving no realistic path forward, since the timetables are practically set in stone. A group could legitimately veer off course maybe 20%, and the solution for running into a brick wall can only be to run into it again at a different angle.

The tread marks, foot prints, cuts and bruises on the brick wall are what's left out of the project, and not what's left in.

Revised Project Plan and Defense of Changes

The original project plan, in this author's opinion, seriously mis-estimates complexity of certain tasks while being insufficiently defined in other areas. The disregarding of dependencies makes targets hard to follow, and contributes a cascading effect. This was partially due to under-preparation, but also partially expected because we lacked experience.

Therefore, changes were necessary to

(a) Realign length of certain tasks to team ability.

(b) Provide more detail on expectation of certain deliverables and reduce repetition.

(c) Distribute points among tasks necessary in development rather than focusing on presentation spectacles.

(d) Correct pacing and difficulty in light team ability, and cohesion.

(e) Reflect better understanding of realistic goals.

Week 1:

Task	Deliverable	Score	
		Allocated	Achieved
Draw block diagram of components.	Block diagram.	2/10	
Order components.	Receipt.	3/10	
Wiring schematic.	Schematic.	3/10	
Complete machine shop orientation	Signed paper	2/10	
Total		10/10	/10

Week 2:

Task	Deliverable	Score	
		Allocated	Achieved
Design car in CAD (1st iteration)	CAD layout.	4/10	
Receive and check BOM.	Checklist of parts.	1/10	
Design car in CAD (2nd iteration)	Final schematic	4/10	
Begin machining parts	Finished car	1/10	
Total		10/10	/10

Week 3:

Task	Deliverable	Score	
		Allocated	Achieved
Calibrate sensors	Calibrated sensors	2/10	
Soldering	Soldered perf board	2/10	
Machine car parts	Refine unwanted parts	2/10	
Wire car together	Built robot.	4/10	
Total		10/10	/10

Week 4:

Task	Deliverable	Score	
		Allocated	Achieved
Wire up car	Functioning car	4/10	
Calibrate sensors	Std. dev. before/after sensors	1/10	
Covariance matrix	Covariance matrix	1/10	
Color sensing	Measured values	4/10	
Total		10/10	/10

Week 5:

Task	Deliverable	Score	
		Allocated	Achieved
Create robot network	Code	5/10	
Setup course for line-following	Demonstration	2/10	
Prepare track with appropriate material	Photos	1/10	
Document progress	Documentation	2/10	
Total		10/10	/10

Week 6:

Task	Deliverable	Score	
		Allocated	Achieved
Develop protocol for robot coordination	Demonstration	7/10	
Document progress	Documentation	3/10	
		/10	
		/10	
Total		10/10	/10

Week 7:

Task	Deliverable	Score	
		Allocated	Achieved
Sense relative fabric position	Sensor data	7/10	
Document progress	Documentation	3/10	
		/10	
		/10	
Total		10/10	/10

Week 8:

Task	Deliverable	Score	
		Allocated	Achieved
Test arm strength	Pictures/video	4/10	
Grasping fabric	Demonstration	2/10	
Test general robot movement	Demonstration	3/10	
		/10	
Total		10/10	/10

Week 9:

Task	Deliverable	Score	
		Allocated	Achieved
Grab fabric simultaneously	Demonstration	2/10	
Synchronous folding	Demonstration	3/10	
Synchronize car speeds	Sensor data	3/10	
Document progress	Documentation	2/10	
Total		10/10	/10

Week 10:

Task	Deliverable	Score	
		Allocated	Achieved
Define starting position	Demonstration	4/10	
Optimize car movements	Demonstration	2/10	
Consistency report	Demonstration and data	2/10	
Document progress	Documentation	2/10	
Total		10/10	/10

Finals Week:

Task	Deliverable	Score	
		Allocated	Achieved
Lab report	Lab report.	10/10	
		/10	
		/10	
		/10	
Total		10/10	/10

Foresight vs Contribution

Despite improved communication at this point, there was still disagreement on complexity and dependencies of certain designs. For instance, more foresight could have accounted for deficiencies in grasping arm design; however, another strain of thought was that a faulty product was better than an unfinished one. Thus, negotiation of the target was an issue. Since there is no team leader, there was no settlement per se. Choices were made by members with their first foot in the door.

Parts of the Final Design

Final design incorporated primarily 3D printed parts, besides the wheels, and hit a target of within 100% over budget. However, not all purchased parts made it into the design, so the overall budget was larger. Grasping arm failed to lift fabric sufficiently to make a fold, so a creative solution involved making a secondary clamp out of the upper jaw, such that the arm clamps up instead of clamping down. The second jaw uses a prior iteration of the arm. A problem of the fabric slipping off the clamp was solved with some dough. It left no residue as per our tests and satisfies usage conditions.

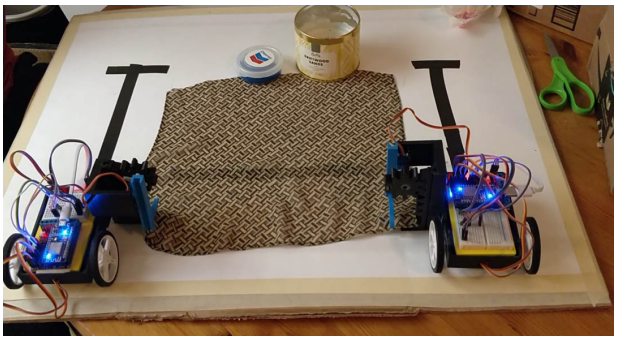

Figure 19: Test run of our final robot.

Minor sizing issues caused us to have to dremel the chassis to make room.

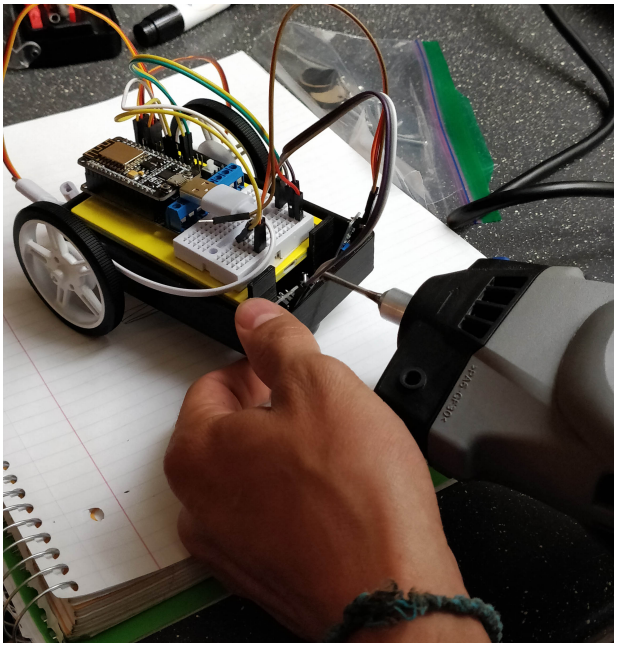

Figure 20: Making room.

Explanation of Code

When initially booted, the master car connects to the WiFi access point with the strongest signal it can detect from a list of known access points. After connecting, it starts a HTTP web server on port 80 and a Web Sockets server on port 81. For the slave car, after the initial boot, it too connects to WiFi access point with the strongest signal which should be the same one the master car is connected to. After connecting to the WiFi, it will immediately attempt to connect to the master car Web Sockets server on port 81. To do this, the local IP of the master car must be known beforehand and hard-coded in the slave car code due to some issues explained later.

Assuming both cars were able to boot successfully and connect to each other, then the protocol for folding the fabric goes as follows. After the slave car initially connects to the master car, it notifies the master car that it is ready to begin. The master car then waits for a request on the HTTP server to begin the folding process. If the local IP address is, for example, 192.168.0.13, then by connecting to http://192.168.0.13/start from a browser, we can begin the folding process. When the request is handled, the master car notifies the slave car that it is also ready to begin. When both cars are ready, they begin whatever state they are on. When one finishes the task for that state, it goes back into standby and lets the other car know that it is ready to continue onto the next task, and this continues until the last state at which point both just sit in a standby state. To end and reset the cars to starting position, we just connect to http://192.168.0.13/end. Using a protocol like this ensures that both cars begin each distinct task at or near the same the time. This is crucial for a task like folding where each corner on one side should lift and fold over simultaneously to avoid unwanted side effects.

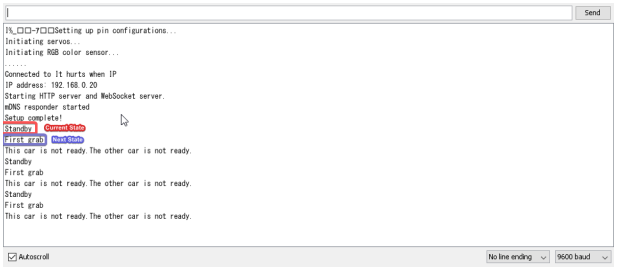

Figure 21: Code of robots synchronizing.

After checking and initializing each component on the car (IR sensors, servos, etc.), It connects to WiFi and outputs the local IP address. It then starts the HTTP server and Web Sockets server and goes into a standby mode waiting to begin.

One downside with this implementation is

that one, you have to keep a list of WiFi access points that you want consider connecting to, and any additions or removals would require you to upload a whole new sketch to the microcontroller. This removes the ability for widespread use, since whenever youre in a new area, youd have to reconfigure both cars. The second downside is that you need to keep track of the local IP address and every time it changes (which can happen even on the same access point when not offline for a decent amount of time), you need to update the slave car with the new local IP address.

Initially, we considered having the master car set up its own access point to which the WiFi name, password, and local IP of the master car could be configured and constant. This way, regardless of where youre at or how often you use the cars, it would always work. However, it seemed the ESP-8266 was unable to simultaneously host an access point and a Web Sockets server. With this setup, whenever the slave car attempted to connect to the master car, it would continuously disconnect and try reconnecting while the master car showed no sign of it ever connecting.

What would we change if we could do it over?

Although some might still hold on to dreams of a CNC, others might suggest that flaws in our approach are multifaceted. Success of any team might be predicated on structure: team and project. Successful leadership gives clear direction might bring out the best quality in each member. However, lack of leadership devolves a team to the least common denominator. The strengths of any single member cannot be effectively utilized over disagreement. Project structure might be whether each member works on separate tasks, or together before moving on to the next task. There are points in a project which tends toward parallelization. Other points tie together various aspects. The fundamental skill in achieving successful structure might not be related specifically to robotics, but to communication. This is something that I lack, and would suggest that other members of the team could work on.

Additionally, we might have assumed too much in the specification and planning phase of this project. In defining original scope, we focused entirely on algorithmic and state estimation factors, and assumed necessary hardware to be available. We assumed spare metal pieces would be inexpensive, or that any kind of part could be 3D printed, without paying attention to the market and limitations.

A change, I suggest, would be to spend more time in designing the grasping arm, with more abstract deliverables, rather than focusing on discrete 3D printed parts or demonstration videos.

We might have also looked more into laser machining or other sourcing of parts, particularly for the grasping arm. This forces us to relax budget constraints, which we ended up going over anyways.

What would we do next?

If budget constraints could be loosened, rack and pinion sets could decouple the grasping arm from the chassis at a smaller footprint than 3D printing, allowing implementation of the lift. Our increased experience of what is necessary for the arm would help in finding another way to fabricate the arm. The 3D printing, however, suffices, provided there are some other changes. We could use a larger microcontroller capable of delivering more power to enhance actuation. A longer term goal might use an overhead camera and machine learning to aid generality; however, this would not be considered until a sufficient grasping arm could be designed. More importantly, better team structure would be required. I would suggest that members take turns assuming leadership role.

Conclusions

Our robotics project is baseline successful, with overall mixed results. However, our educational project is very successful. We, each of us, learned a lot about designing projects to be successful, choosing appropriate metrics, measuring complexity of subtasks, streamlining projects to minimize dependencies, and most importantly, the usefulness of team composition and communication. We realize that in a work or research setting, we cannot always choose who we work with, so a diverse toolbox in working with different kinds of people is instrumental to our success, whether in robotics or any field. And this class has taught us that.

Bibliography

[1] "Laundroid is an Awesome Laundry-Folding Robot". *Electronic House*, 19 Jan. 2016.

[2] Parker, Lynne E. "Multiple Mobile Robot Systems". *Springer Handbook of Robotics*: p921-941, Jan. 2008.

[3] Ravven, Wallace. "Deep Learning: A Giant Step for Robots". *Berkeley Research*, 22 Feb. 2016.

Appendix A: Bill of Materials

	Vendor	Vendor Part #	Manufacturer	Part #	Use	Description	Quantity	Unit Price	Total Cost
Adafruit		2442		2442	Rotary Motion	FS90R continuous servo	4	7.5	30
Adafruit		169		169	Rotary Motion	SG90 servo motor	2	5.95	11.9
Adafruit		SKU 113990105		SKU 113990105	Processing/Computing	EP8S266 NodeMCU	2	0	0
Mouser		782-TCRT5000L	Texas Advanced Opto	TCS34725	sensor's measured light intensity is outside of a giver	Color sensor TCS34726	6	1.21	7.26
Digi-Key		987-1188-ND	TT Electronics/BI	EN11-HSM1AF15	Motor Functionality	Rotary encoders		1.22	7.32
Banana Robotics		BR010266	Solarbotics	Wheels	Necessary to traverse folding table		8	3.95	31.6
Pololu		961	Pololu	961	Tape Detection	Faster line following	2	9.95	19.9
UCLA Store		N/A	Gorilla Glue	-	Adhese Wheels to Servomotors	Super Glue; Adhesive	1	4.49	4.72
Amazon		1395391	Loctite	1395391	Glue Chassis and Servomotors together	Adhesive	1	6.65	7.22
Amazon		B00F5PNISQ	Cofufu	B00F5PNISQ	Act as passive wheel; support	Wheels, passive	1	6.5	6.5
Pololu		1429	Pololu	1429	Wheels To Move Cars	Wheels, active	2	8.49	16.98
Pololu		953	Pololu	953	Passive wheel to support chassis	Wheels, passive	2	1.99	4
Ralph's Supermarket		-	ArtSkills	-	white background for "racetrack"	White Board	1	5	5
Target		-	3M	-	Electrical tape to provide track for clothes perimeter	Electrical Tape	1	3.49	3.49
							Total:		155.89

Appendix B: ESP-8266 Code

```
/*
* 1.) Master starts server, slave connects
* 2.) For each state, both signify they are ready before they do anything. When
*      one car is ready before the other, that car will just idly wait in standby.
* 3.) The progression of states is as follows:
*       - STANDBY
*       - FIRST_GRAB,
*       - FIRST_FOLD,
*       - FIRST_FOLD_END,
*       - SECOND_PHASE_SETUP,
*       - SECOND_GRAB,
*       - SECOND_FOLD,
*       - SECOND_FOLD_END,
*       - STANDBY
*/

#define MASTER 0
#define CAR_SPEED_FACTOR 0.25f
#define MASTER_IP "172.20.10.4"

#include <ESP8266WiFiMulti.h>
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <WebSockets.h>
#include <Adafruit_TCS34725.h>
#include <Servo.h>

#define SERVO_RIGHT_PIN      D0
#define SERVO_LEFT_PIN       D3
#define SERVO_GRIP_PIN       D4
#define IRSENSOR_RIGHT_PIN   D7
#define IRSENSOR_LEFT_PIN    D8
#define GRIP_ANGLE           60
#define REST_ANGLE           120

typedef enum State
{
  FIRST_GRAB,
  FIRST_FOLD,
  FIRST_FOLD_END,
  SECOND_PHASE_SETUP,
  SECOND_GRAB,
  SECOND_FOLD,
  SECOND_FOLD_END,
  STANDBY
} State;


// SHARED VARS + FUNCTIONS
ESP8266WiFiMulti WiFiMulti;
Servo servoRightWheel;
Servo servoLeftWheel;
Servo servoGrip;
Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_700MS,
  TCS34725_GAIN_1X);

```

```
56  State currentState;
57  State nextState;
58  bool isReady, isOtherReady;
59  int numIntersections;
60
61  void setupPins();
62  void initComponents();
63  void connectToWifi();
64
65  void setReady();
66  State getNextState(State currentState);
67
68  void readSensorValues(int sensorValues[]);
69  void driveWheels(byte leftVal, byte rightVal);
70  void stopServos();
71
72  void driveForward(float speedFactor);
73  void turn(bool clockwise, float speedFactor);
74  void openGrip();
75  void closeGrip();
76
77  void grabFabric();
78  void followLine(int requiredNumIntersections);
79  void ungrabFabric();
80
81
82  // MASTER VARS + FUNCTIONS
83  #if MASTER
84    #include <WebSocketsServer.h>
85    #include <ESP8266mDNS.h>
86    #define GRIP_ANGLE          105
87    #define REST_ANGLE          60
88
89    ESP8266WebServer server(80);
90    WebSocketsServer webSocketServer = WebSocketsServer(81);
91
92    void initServers();
93    void handleStartRequest();
94    void handleStopRequest();
95    void handleNotFound();
96    void webSocketServerEvent(uint8_t num, WStype_t type, uint8_t * payload, size_t
      length);
97
98
99  // SLAVE FUNCTIONS
100 #else
101   #include <WebSocketsClient.h>
102
103   #define GRIP_ANGLE        70
104   #define REST_ANGLE        30
105
106   bool connectedToMaster;
107   WebSocketsClient webSocketClient;
108
109   void connectToMaster();
110   void webSocketClientEvent(WStype_t type, uint8_t * payload, size_t length);
111 #endif
112
113 // DEBUG ONLY
```

```
114  void dumpInfo()
115  {
116    if (isReady)
117      Serial.print("This car is ready. ");
118    else
119      Serial.print("This car is not ready.");
120
121    if (isOtherReady)
122      Serial.println("The other car is ready.");
123    else
124      Serial.println("The other car is not ready.");
125  }
126
127
128  // DEBUG ONLY
129  void printState(State state)
130  {
131    switch (state)
132    {
133    case FIRST_GRAB:
134      Serial.println("First grab");
135      break;
136
137    case FIRST_FOLD:
138      Serial.println("First fold");
139      break;
140
141    case FIRST_FOLD_END:
142      Serial.println("First fold end");
143      break;
144
145    case SECOND_PHASE_SETUP:
146      Serial.println("second phase setup");
147      break;
148
149    case SECOND_GRAB:
150      Serial.println("Second grab");
151      break;
152
153    case SECOND_FOLD:
154      Serial.println("Second fold");
155      break;
156
157    case SECOND_FOLD_END:
158      Serial.println("Second fold end");
159      break;
160
161    case STANDBY:
162      Serial.println("Standby");
163      delay(100);
164      break;
165
166    default:
167      Serial.println("default state...idk why this was called.");
168      break;
169    }
170  }
171
172  void setup()
```

```
{
  Serial.begin(9600);
  //Serial.setDebugOutput(true);

  setupPins();
  initComponents();
  connectToWifi();

  isReady = false;
  isOtherReady = false;
  currentState = STANDBY;
  nextState = FIRST_GRAB;
  numIntersections = 0;

  #if MASTER
    initServers();
  #else
    connectToMaster();
  #endif

  Serial.println("Setup complete!");
  delay(1000);
}

unsigned long lastPrintTime = 0;
void loop()
{
  #if MASTER
    server.handleClient();
    webSocketServer.loop();

  #else
    webSocketClient.loop();

    if (!connectedToMaster)
      return;

  #endif


  // print the state of the car every 5 seconds
  if (millis() - lastPrintTime >= 5000)
  {
    printState(currentState);
    printState(nextState);

    dumpInfo();

    lastPrintTime = millis();
  }

  switch (currentState)
  {
  case FIRST_GRAB:
    grabFabric();
    break;

  case SECOND_GRAB:
    //grabFabric();
```

```
232       break;
233
234     case FIRST_FOLD:
235       #if MASTER
236         followLine(1);
237       #else
238         //followLine(2);
239         followLine(1);
240       #endif
241
242     case FIRST_FOLD_END:
243     case SECOND_FOLD_END:
244       //ungrabFabric();
245       break;
246
247     case SECOND_PHASE_SETUP:
248       break;
249       // The slave car will need to hit 3 intersections before stopping
250       #if MASTER
251         followLine(2);
252       #else
253         followLine(3);
254       #endif
255         break;
256
257     case SECOND_FOLD:
258       followLine(2);
259       break;
260
261     case STANDBY:
262       break;
263
264     default:
265       Serial.println("default state...idk why this was called.");
266       break;
267     }
268   }
269
270   ////////////////////////////// SETUP/INITIALIZATION METHODS
      //////////////////////////////
271   void setupPins()
272   {
273   Serial.println("Setting up pin configurations...");
274
275   pinMode(SERVO_RIGHT_PIN, OUTPUT);
276   pinMode(SERVO_LEFT_PIN, OUTPUT);
277   pinMode(SERVO_GRIP_PIN, OUTPUT);
278
279   pinMode(IRSENSOR_RIGHT_PIN, INPUT);
280   pinMode(IRSENSOR_LEFT_PIN, INPUT);
281   }
282
283   void initComponents()
284   {
285   Serial.println("Initiating servos...");
286
287   servoRightWheel.attach(SERVO_RIGHT_PIN);
288   servoLeftWheel.attach(SERVO_LEFT_PIN);
289   servoGrip.attach(SERVO_GRIP_PIN);
```

```
290
291   Serial.println("Initiating RGB color sensor...");
292   // if (!tcs.begin())
293   //{
294   //Serial.println("TCS34725 RGB sensor not found.");
295   //while(1);
296   //}
297
298   // Make sure wheels arent moving
299   driveWheels(90, 90);
300
301   // Open the grip
302   servoGrip.write(REST_ANGLE);
303   }
304
305   void connectToWifi()
306   {
307   WiFi.mode(WIFI_STA);
308
309   // Any new wifi pass combos go here
310   WiFiMulti.addAP("It hurts when IP", "ozzy3333");
311   WiFiMulti.addAP("E-Chuta", "howrude66");
312   WiFiMulti.addAP("NBB", "Xray72mare0336icon");
313
314   // Wait for the Wi-Fi to connect: scan for Wi-Fi networks, and connect to the
        strongest of the networks above
315   while (WiFiMulti.run() != WL_CONNECTED)
316   {
317   delay(1000);
318   Serial.print('.');
319   }
320
321   // Tell us what network we're connected to
322   Serial.println();
323   Serial.print("Connected to ");
324   Serial.println(WiFi.SSID());
325   Serial.print("IP address: ");
326   Serial.println(WiFi.localIP());
327   }
328
329   void setReady()
330   {
331   isReady = true;
332   currentState = STANDBY;
333
334   #if MASTER
335   webSocketServer.broadcastTXT("ready");
336   #else
337   webSocketClient.sendTXT("ready");
338   #endif
339
340   if (isOtherReady)
341   {
342   currentState = nextState;
343   nextState = getNextState(currentState);
344
345   isReady = false;
346   isOtherReady = false;
347   }
```

```
348 }
349
350 State getNextState(State currentState)
351 {
352 switch (currentState)
353 {
354 case FIRST_GRAB:           return FIRST_FOLD;
355 case FIRST_FOLD:           return FIRST_FOLD_END;
356 case FIRST_FOLD_END:       return SECOND_PHASE_SETUP;
357 case SECOND_PHASE_SETUP:   return SECOND_GRAB;
358 case SECOND_GRAB:          return SECOND_FOLD;
359 case SECOND_FOLD:          return SECOND_FOLD_END;
360 case SECOND_FOLD_END:      return STANDBY;
361 default:                   return STANDBY;
362 }
363 }
364 ////////////////////////////////////////////////////////
365
366 ////////////////////////////////// SENSOR + ACTUATOR METHODS
      //////////////////////////////////
367 void readSensorValues(int sensorValues[])
368 {
369 sensorValues[0] = digitalRead(IRSENSOR_LEFT_PIN);
370 sensorValues[1] = digitalRead(IRSENSOR_RIGHT_PIN);
371
372 // read color sensor
373 // read wheel ir sensor
374 }
375
376 void driveWheels(byte leftValue, byte rightValue)
377 {
378 servoLeftWheel.write(leftValue);
379 servoRightWheel.write(rightValue);
380 }
381
382 void stopServos()
383 {
384 servoRightWheel.write(90);
385 servoLeftWheel.write(90);
386 servoGrip.write(REST_ANGLE);
387 }
388
389 void driveForward(float speedFactor)
390 {
391 speedFactor = constrain(speedFactor, 0.0, 1.0);
392 driveWheels(90 * (1 + speedFactor), 90 * (1 - speedFactor));
393 }
394
395 void turn(bool clockwise, float speedFactor)
396 {
397 speedFactor = constrain(speedFactor, 0.0, 1.0);
398 if (clockwise)
399 driveWheels(90 * (1 + speedFactor), 90 * (1 + speedFactor));
400 else
401 driveWheels(90 * (1 - speedFactor), 90 * (1 - speedFactor));
402 }
403
404 void openGrip()
405 {
```

```
servoGrip.write(GRIP_ANGLE);
}

void closeGrip()
{
servoGrip.write(0);
}

void grabFabric()
{
Serial.println("Grabbing fabric...");
servoGrip.write(GRIP_ANGLE);

delay(3000);
setReady();
}

//bool isFollowingLine = false;
void followLine(int requiredNumIntersections)
{
int sensorValues[2];
readSensorValues(sensorValues);

int leftIRVal = sensorValues[0];
int rightIRVal = sensorValues[1];

if (leftIRVal == HIGH && rightIRVal == HIGH)
{
numIntersections += 1;
if (numIntersections >= requiredNumIntersections)
{
//isFollowingLine = false;
numIntersections = 0;
stopServos();

// This check is not needed if the track just loops around instead of trying to do
   a 180
//      #if !(MASTER)
//        if (currentState == SECOND_STAGE_SETUP)
//
//      #endif

// This car is now ready to start next state.
setReady();
}
}
else if (leftIRVal == HIGH && rightIRVal == LOW)
{
turn(false, CAR_SPEED_FACTOR);
delay(100);
}
else if (leftIRVal == LOW && rightIRVal == HIGH)
{
turn(true, CAR_SPEED_FACTOR);
delay(100);
}
else
driveForward(CAR_SPEED_FACTOR);
}
```

```
464
465 void ungrabFabric()
466 {
467 Serial.println("Letting go of fabric.");
468 servoGrip.write(REST_ANGLE);
469 delay(3000);
470
471 setReady();
472 }
473 ////////////////////////////////////////////////////////
474
475 #if MASTER
476 ////////////////////////////////// MASTER METHODS//
    ////////////////////////////////
477 void initServers()
478 {
479 Serial.println("Starting HTTP server and WebSocket server.");
480
481 server.on("/start", handleStartRequest);
482 server.on("/stop", handleStopRequest);
483 server.onNotFound(handleNotFound);
484 server.begin();
485
486 delay(1000);
487 webSocketServer.begin();
488 webSocketServer.onEvent(webSocketServerEvent);
489
490 // Start the mDNS responder for esp8266fb.local
491 if (!MDNS.begin("esp8266fb"))
492 Serial.println("Error setting up MDNS responder!");
493 else
494 {
495 Serial.println("mDNS responder started");
496 MDNS.addService("http", "tcp", 81);
497 MDNS.addService("http", "tcp", 80);
498 }
499 }
500
501 void handleStartRequest()
502 {
503 // Send repsonse back to http client
504 server.send(200, "text/plain", "OK");
505
506 Serial.println("Start request");
507 Serial.flush();
508
509 // Put car into standby and wait notify slave car that we're ready to begin
510 currentState = STANDBY;
511 nextState = FIRST_GRAB;
512 setReady();
513
514 //driveForward();
515 }
516
517 void handleStopRequest()
518 {
519 // Send repsonse back to http client
520 server.send(200, "text/plain", "OK");
521
```

```
522   Serial.println("Stop request");
523   Serial.flush();
524
525   stopServos();
526   isOtherReady = false;
527   currentState = STANDBY;
528   nextState = FIRST_GRAB;
529   numIntersections = 0;
530
531   webSocketServer.broadcastTXT("stop", strlen("stop"));
532   }
533
534   void handleNotFound()
535   {
536   server.send(404, "text/plain", "File not found.");
537   }
538
539   void webSocketServerEvent(uint8_t num, WStype_t type, uint8_t * payload, size_t
        len)
540   {
541   switch(type)
542   {
543   case WStype_DISCONNECTED:
544   Serial.printf("[%u] Disconnected!", num);
545   Serial.println();
546
547   stopServos();
548
549   isOtherReady = false;
550   break;
551
552   case WStype_CONNECTED:
553   {
554   IPAddress ip = webSocketServer.remoteIP(num);
555   Serial.printf("[%u] Connected from %d.%d.%d.%d url: %s", num, ip[0], ip[1], ip[2],
        ip[3], payload);
556   Serial.println();
557   }
558   break;
559
560   case WStype_TEXT:
561   Serial.printf("[%u] get Text: %s", num, payload);
562   Serial.println();
563
564   if (strcmp((char *) payload, "ready") == 0)
565   {
566   isOtherReady = true;
567
568   if (isReady)
569   {
570   currentState = nextState;
571   nextState = getNextState(currentState);
572   numIntersections = 0;
573
574   isReady = false;
575   isOtherReady = false;
576   }
577   }
578   break;
```

```
579
580 case WStype_BIN:
581 Serial.printf("[%u] get binary length: %u", num, len);
582 Serial.println();
583 hexdump(payload, len);
584 break;
585
586 default:
587 Serial.printf("Invalid WStype [%d]", type);
588 Serial.println();
589
590 break;
591 }
592
593 Serial.flush();
594 }
595 ////////////////////////////////////////////////////////////
596
597 #else
598 ////////////////////////////////// SLAVE METHODS
      //////////////////////////////////
599 void connectToMaster()
600 {
601
602 webSocketClient.begin(MASTER_IP, 81);
603 webSocketClient.onEvent(webSocketClientEvent);
604 webSocketClient.setReconnectInterval(5000);
605 }
606
607 void webSocketClientEvent(WStype_t type, uint8_t * payload, size_t len)
608 {
609 switch(type)
610 {
611 case WStype_DISCONNECTED:
612 Serial.println("Disconnected");
613 connectedToMaster = false;
614 isOtherReady = false;
615
616 stopServos();
617
618 break;
619
620 case WStype_CONNECTED: {
621 Serial.printf("Connected to %s", payload);
622 Serial.println();
623
624 connectedToMaster = true;
625 setReady();
626 }
627 break;
628
629 case WStype_TEXT:
630 Serial.printf("Text: %s", payload);
631 Serial.println();
632
633 if (strcmp((char *) payload, "ready") == 0)
634 {
635 isOtherReady = true;
636
```

```
637   if (isReady)
638   {
639   currentState = nextState;
640   nextState = getNextState(currentState);
641   numIntersections = 0;
642
643   isReady = false;
644   isOtherReady = false;
645   }
646   }
647   else if (strcmp((char*) payload, "stop") == 0)
648   {
649   stopServos();
650
651   isOtherReady = false;
652   currentState = STANDBY;
653   nextState = FIRST_GRAB;
654   numIntersections = 0;
655   setReady();
656   }
657
658   break;
659
660   case WStype_BIN:
661   Serial.printf("Binary data length: %u", len);
662   Serial.println();
663
664  hexdump(payload, len);
665   break;
666   }
667   }
668   ////////////////////////////////////////////////////////////
669
670  #endif
671
672
```

Listing 1: Robot movement and intercommunication.