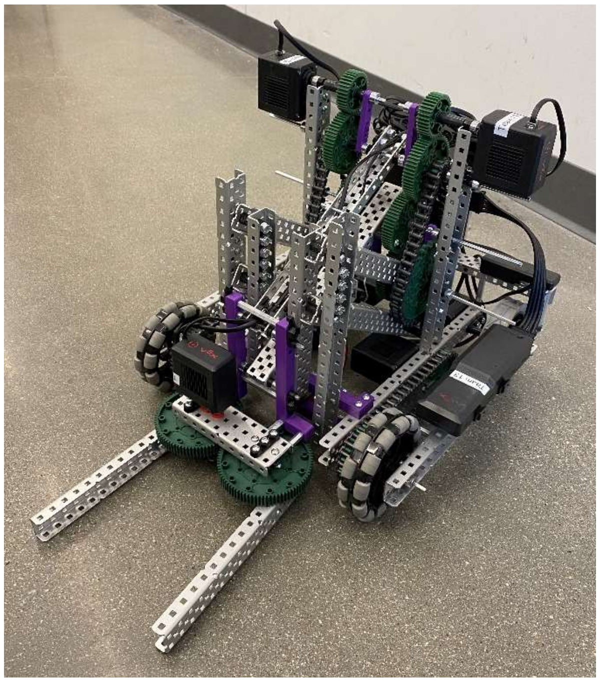
RBE 1001 Team 13 C2022

Flynn Duniho, Evan Carmody, Renata Kaplan



Division of Labor

Flynn Duniho 33.3%	Worked on drivetrain and collection/lifting, main head of autonomous and tele-op programs and source control for code
Evan Carmody 33.3% Evan Carmody	Designed and built the drivetrain, worked on the lifting/collection mechanisms, worked on autonomous and tele-op programs
Renata Kaplan 33.3%	Designed and built the collection/lifting mechanisms, worked on autonomous and tele-op programs

Introduction and Problem Statement

Our team was tasked with designing and building a robot capable of delivering pizzas to various residence halls around a pre-defined “campus” arena. Delivering pizzas and performing various other tasks give our robot and team points, which we are trying to maximize within the 10-minute time slot. These additional tasks include pushing a large wooden 10.4 lb block (called “herding Gompei”), picking up smaller square blocks (called “pizzas”) and placing them into multilevel structures (called “dorms”), placing a pizza in each level of a dorm (achieving this is called getting a “happy dorm”) and then delivering a special ”golden pizza”, and autonomously navigating the playing field, driving up a ramp, and delivering pizzas.

Pizza delivery is our main priority, but we hope to create a robot capable of demonstrating the other tasks, to maximize our points scored. This task is made difficult due to various constraints, including but not limited to:

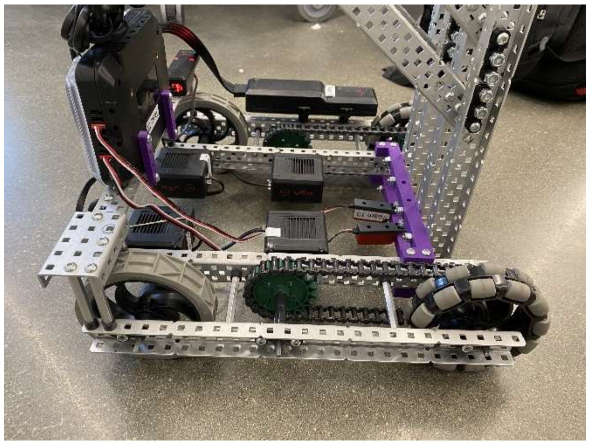
1. The size constraint of 18.75” by 15.25” by 15.25”
2. A weight limit of 10.0 lbs
3. The fact that pizzas only count as scored if at least 50% of it is in the dorm
4. Our limited access to parts for construction
5. The requirement that our robot must climb a ramp to access the arena
6. The limitation that our robot can only possess one pizza at a time

We endeavor to create a robot that remains within the size and weight constraints when the match begins, can deliver pizzas to all floors of the dorms, achieve a happy dorm within the first 5 minutes, autonomously drive up the ramp, pick up pizzas from both the floor and from player load, and successfully herd Gompei to our side.

System Design

The main objectives we needed to achieve were driving up the ramp, delivering pizzas, and herding Gompei. The ramp necessitates high traction for our robot so we don't slip down the ramp, as well as higher torque so we can drive on the incline. We needed a simple and reliable claw and a fast-lifting mechanism capable of reaching all the floors of the dorms. Due to the fast paced, timed nature of the competition, we believed it was essential that both our drivetrain and pizza manipulation mechanisms were fast and allowed us to quickly navigate the field and deliver pizzas.

Drive Train



Our drivetrain consists of a pair of omni-wheels in the front and a pair of standard wheels in the back. It's a good compromise between the freedom of movement given by using four omni wheels and the traction afforded by exclusively using standard wheels. For the front of the robot to house our lifting and claw mechanism, we needed it to be relatively open. To accomplish this, we positioned all four drive motors near the rear of the robot and chained two motors to the front Omni-directional wheels. We wanted a one-to-one gear ratio on the drivetrain for the speed, but we needed enough power to drive up the ramp. By driving all four wheels, we have enough power to drive up the ramp, and enough speed to travel between the tasks, quickly moving between the dorms and pizzerias.

Claw

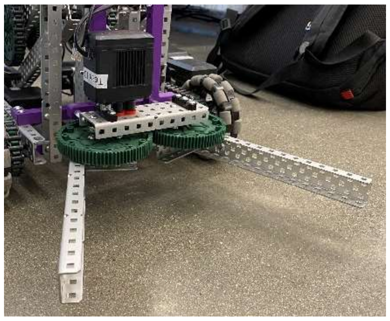


Fig. Claw mechanism attached to lifter

For primary manipulation of the pizzas, we ultimately ended up going with a claw. It is relatively simple, and we decided it would be more reliable than a complex intake design. The claw is made up of two meshing 84 tooth gears powered by a single motor. A pair of C-channels then attach to the gears, serving as the grippers for the claw. It's a fairly simple design, but it's quite compact, and it's incredibly reliable. Having a claw allows us to easily pick up from the floor, which gives us a 1.5* bonus to the pizzas we deliver in driver control.

Lifting Mechanism

To lift the collected pizzas to the different floors of the dorm, our final design made use of a double reverse four-bar lifting mechanism. This makes the robot very compact vertically and keeps the claw moving mostly parallel to the floor. While a standard reverse double four-bar lifter should be aligned so both sets of horizontal bars can be parallel with the floor at a point, we found running them slightly offset enabled us to reach the floor, which we were not able to do with them completely parallel. While this does cause the claw to move at a slight arc near the top, it makes no functional difference as the change in angle is negligible when delivering pizza. While testing the design, we noticed that the axles were under a large amount of stress and bent in opposite directions under the weight of the arms. To solve this, we designed and printed links (fundamentally 4, 5, and 6 hole flat bearings) connecting the different axels to keep everything in line and reduce bending stress from the high amounts of torque. We use a series of gears and sprockets with chains to increase torque, as lifting the claw and keeping it up require a lot of force. In total, the gear ratio from the driven axle to the channels of the lifting mechanism is 3:37. In addition to gearing down, the chains also provide upwards force on the axles, pushing the two sets of gears together and reducing skipping, which was a consistent issue before. Having the motor mounted on the second parallel bar of the lifter instead of the mount allowed us more

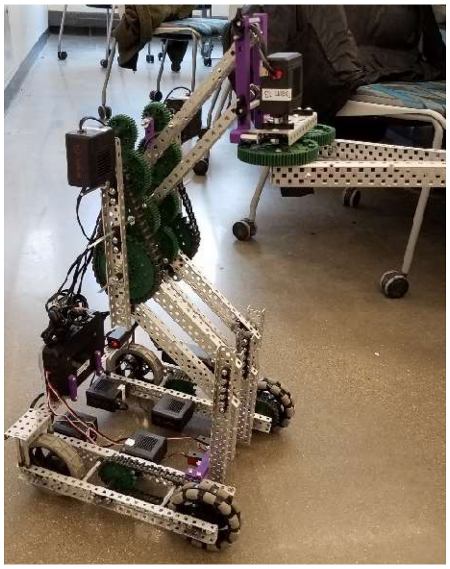


Fig. The robot fully extended

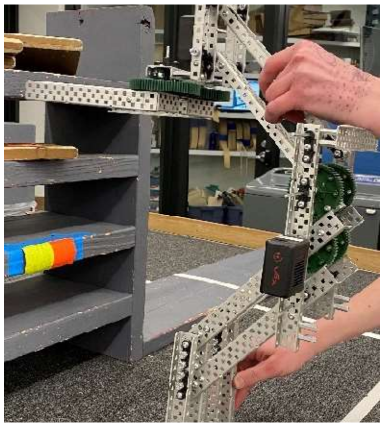


Fig. Early iteration of the lifter

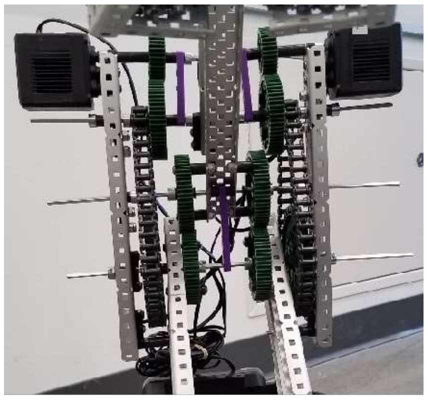


Fig. Gearing system fully extended

space to gear down and reinforce the system. Overall, we found the reverse double four-bar lifter to be reliable, fast, and easy to use and incorporate with our other subsystems.

Solutions and Justifications

The robot must be able to deliver to the top floor of the dorms, 42cm off the ground within 5 seconds:

As an initial design for a lifting mechanism, we started with a rack and pinion system, two pinion gears moving up along a multistage rack. Because of the slow lifting/lowering speed, difficulty to support/stabilize, and low maximum lifting height, we decided to switch to a four-bar lifter. We tried a single four-bar lifter, but we were limited by parts and couldn't make it the size we would have liked while still reaching all dorm floors. After this iteration, we switched to the final double reverse four-bar lifter. This turned out to be a much better solution, because the starting vertical height is much shorter than a single four-bar lifter, allowing our claw to reach the ground and pick up ground floor pizzas. The claw doesn't move in an arc, yet we still have the same speed as a single four-bar lifter.

The robot must be able to pick up the pizzas quickly, and should only unintentionally drop the pizzas no more than 5% of the time:

During the early experimentation stage, we found that a one-inch-wide C-channels almost perfectly fit the height of the pizzas. Because of this, our claw has pretty much stayed the same, as we happened upon a very flexible, reliable, and fast design. It is simple yet can pick up pizzas no matter the rotation. Even if half the pizza is hanging out of our claw, it's still very reliable and doesn't drop the pizzas accidentally. We use two large gears for the opposing motion needed in the grippers. And we use C-channels for the grippers since they fit almost perfectly around the pizzas. The grip is secure and consistent.

The robot must be able to drive up a wooden ramp at a 30 degree angle:

We tested a few different drivetrains until we settled on the omni wheel in front, standard wheel in back combination that we use now. At first we tried to make a 6-wheel robot, with omni wheels on the outside and standard wheels in the center. We had issues with this because the standard wheels were ever so slightly smaller in diameter than the omni wheels. This meant that the center standard wheels didn't touch the ground, even when wrapped in rubber bands. The next configuration we tried was using four omni wheels. This was nice because the pivot point of the robot was in the center of the robot, but it wasn't great for tractions. We were barely able to drive up the ramp, and the robot would start slipping back down the ramp if it turned even a little bit sideways. This wasn't great, so we finally decided on standard wheels in the back for traction, and omni wheels in the front for maneuverability.

The robot must be able to push Gompei:

Because we designed for the traction constraints needed to drive up the ramp, we have no problems pushing Gompei.

The robot must be able to navigate the field autonomously:

Since there's an autonomous portion, we wanted to add sensors to aid the positioning of the robot. We are using line sensors on the bottom of the robot to follow the lines marked on the playing field. Since the lines are in all the large areas of the playing field (including the ramp), we can program the robot to correct its course automatically instead of purely relying on turning the motors a certain amount. The lines don't lead straight to the dorms though, so we use dead reckoning for the small distances from between the lines and the dorms. The robot can accurately and consistently navigate to the dorms and correct for any error that may occur.

The robot must be able to deliver pizzas autonomously:

With the line sensors, the robot can navigate accurately to the dorms. We use the motor encoders to lift the claw to exactly the right height to deliver the pizzas, and from there it's a simple matter of driving into the dorm and releasing the pizza.

System Integration

In autonomous, the robot first grips with the claw, grabbing the pizza we place in front of it, then lifts so the claw and pizza don't get caught on the floor of the ramp. Using the line sensors, the robot drives up the ramp, following the black centerline. We know when the robot reaches the top of the ramp because that's when both line sensors detect the dark carpet at the same time. From there we can drive until we find a white line, then drive forwards a predetermined amount using encoders and deliver the pizza into the first floor of the Messenger dorm. Once the robot is in a known location, only a little bit of dead reckoning is needed to move into position to pick up and deliver the pizzas. We need to set the claw to a specific height to deliver the pizzas, so we use the motor encoders in the four-bar motors to lift the claw to the required height. The robot will need to drive over a raised portion of the field, and that can throw off the position of the robot by quite a bit. However, we only need the robot to be fully within the construction zone, as it ends autonomous there. For the driver-controlled portion, we will still use the motor encoders in the four-bar to lift the claw to predetermined heights. The claw will be on a toggle, so much of the repetitive tasks will be automated even when a person is driving the robot.

Performance

We tested the lifting mechanism and found that we could lift from the bottom floor all the way to the top floor in an average of 1.7 seconds. For driving speed, the robot can drive across the field in an average of 3.7 seconds, well meeting our requirements. Any pizza that was properly seated in the claw was never dropped in any of our tests, yielding a 100% success rate. The only time pizzas were dropped was when the pizzas were barely seated, or not seated at all. Autonomous works with a 100% accuracy, thanks to the line sensors we employ for navigation and the flexibility that we built into the robot's path.

During testing, we found running the robot for too long led to the lifting motors overheating. Because of the high amounts of torque needed to keep the lifter up, the motors would give out and we would need to wait for them to cool back down before continuing work. If we were to improve the bot more, finding a way to take stress off the motors would make the bot quite a bit better.

Appendices

For the ramp, the two major tests we needed to pass was for tipping and slipping. We need enough traction to drive up the ramp and we need the center of gravity low enough so that the robot doesn't fall over. During one of the labs, we made a program to calculate the maximum angles for both tipping and slipping given the robot dimensions relative to the center of gravity, and the coefficient of friction.

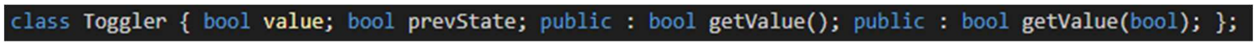
$$\arctan\left(\frac{a}{(a+b)/\mu - h}\right) = \alpha$$

Fig. Derived equation to find the maximum angle without the robot slipping

$$\arctan\left(\frac{a}{h}\right) = \alpha$$

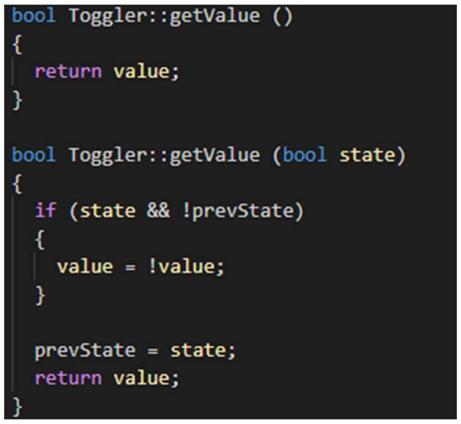
Fig. Derived equation for the maximum angle without the robot tipping over

All of our code was synchronized via Github. This allows us to roll back the code to previous versions if we accidentally break something. It also allows us to easily share code to everyone else on the team. We made sure to keep the code as organized as possible to allow for reusing code when needed. All of the dead reckoning and line following code was pulled out into methods to be reused, as well as the lifting logic to lift to a specific floor. Shown below is an example of a class that we made to handle toggling between two states using a single button.



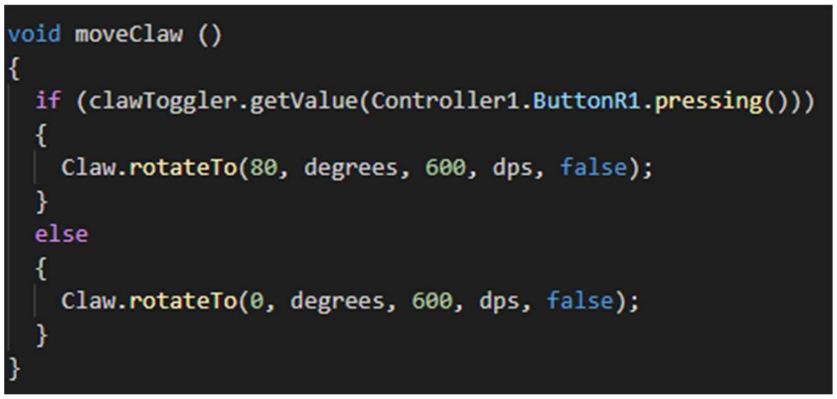
```
class Toggler { bool value; bool prevState; public : bool getValue(); public : bool getValue(bool); };
```

Fig. The Toggler declaration in Util.h



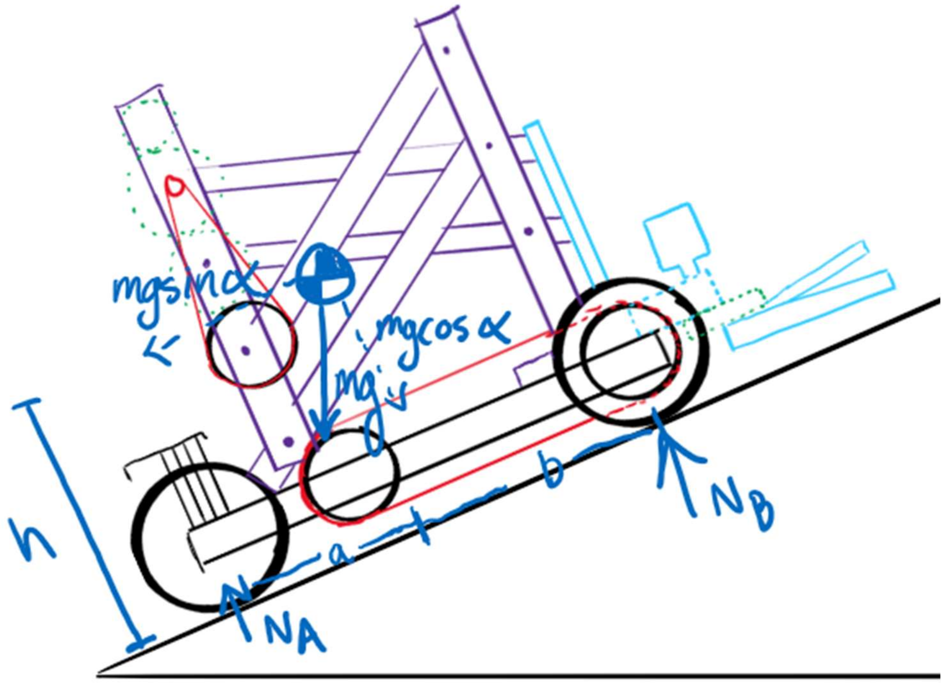
```
bool Toggler::getValue ()
{
  return value;
}

bool Toggler::getValue (bool state)
{
  if (state && !prevState)
  {
    value = !value;
  }

  prevState = state;
  return value;
}
```

Fig. The Toggler implementation in Util.cpp



```
void moveClaw ()
{
  if (clawToggler.getValue(Controller1.ButtonR1.pressing()))
  {
    Claw.rotateTo(80, degrees, 600, dps, false);
  }
  else
  {
    Claw.rotateTo(0, degrees, 600, dps, false);
  }
}
```

Fig. The Toggler class in use for the claw in TeleOp


mgsinα
mgcosα
mg
h
a
b
N_A
N_B