ResQTech

ROBOTICS DIVISION

To: Robert Gettens, Ph.D., Chief Executive Officer

From: Duct Tape and Dreams & #155

Team Member	Role	Report Contribution
Bailey Kinney	Leader	Formatting and data collection
Connor Lord	Recorder	Photos and Rescue Bot
Cooper McConaghy	Consultant	Bridge Bot and Photos
Shadiel Marte	Time Keeper	Bridge Bot

Date: 12/10/2025

Subject: Design History File Bridge and Rescue Bots

Contents

EMPATHIZE & DEFINE

Problem Description

The purpose of this design challenge is to rescue Lego people from lava across various terrains. This terrain includes a speed bump, a gap with lava below, and a small wall. The Lego people are standing on the wall and waiting for rescue. The robot(s) will be remote-controlled and need to be able to navigate across this varied terrain.

Problem Statement

First responders need a set of robots to rescue LEGO people who are stranded beyond various forms of terrain, because it is hazardous for humans to go out there to save the people.

Various pieces of information are needed about the course, including the size of the course and the obstacles along the course. The team needs to further explore the components included in the robot kits. Additionally, the team must gather detailed information about the Lego figures, including their dimensions.

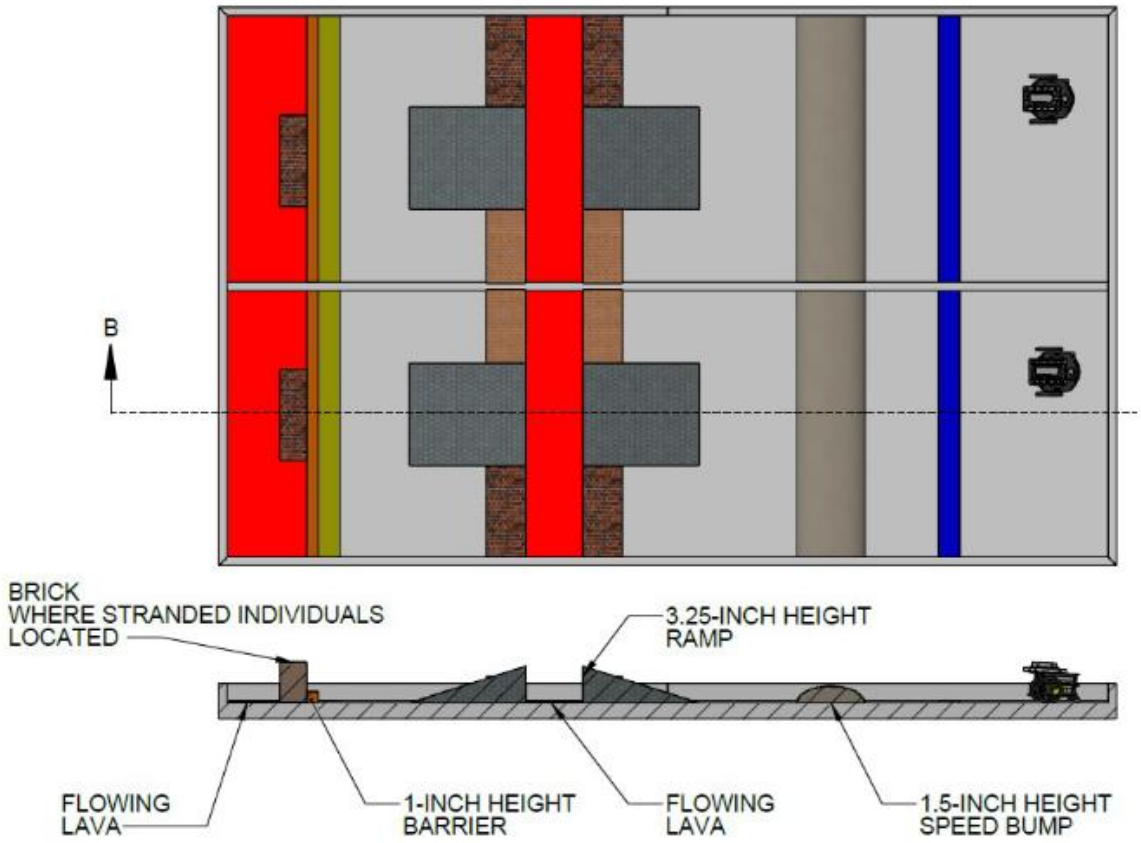
Constraints

The constraints of this design challenge are listed below.

Constraints:

1. Must use a maximum of three robots to save these Lego people from the dangerous terrain.
2. The robots must not touch the lava whatsoever.
3. The robots must scale two obstacles, a 1.5” bump and two 3.25” tall ramps with a gap in between them.

Figure #1: The Course



4. Must only use the electronic parts provided within the robot kits.
5. The amount of time to complete the rescue must remain under four minutes.

Figure #2: Time Scoring

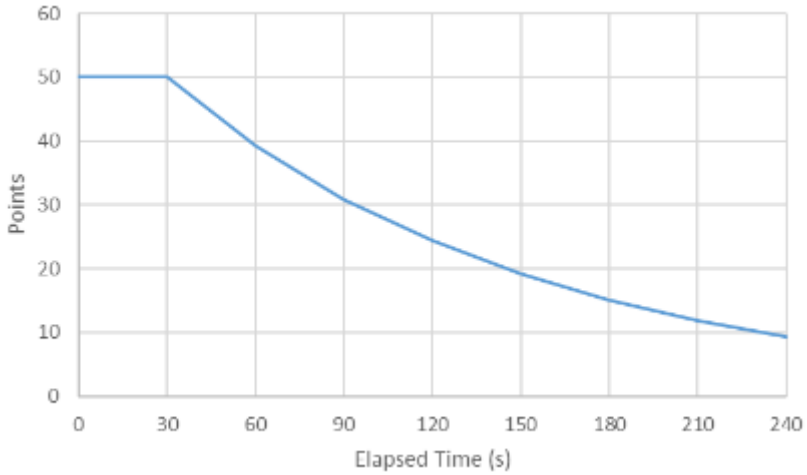


Figure #2 shows the point vs elapsed time curve for the robot competition. This displays which times will result in which amount of points for the final competition.

Criteria for Success

The main keys to success center around efficiency in design and execution.

- Maximize the number of Lego people and Golden Bears saved.
- Minimize the number of robots used.
- Minimize the time taken to complete the task (less complexity).
- Make the solution as simple as possible to navigate.
- Minimize the amount of time it takes to 3D print.
- Minimize the number of attempts over each obstacle.

The number of people and bears saved will be measured by how many people and bears the rescue bot can bring back to the starting point. Efficiency is key in robot use, allowing most parts to be used for the robots by only using two robots, one for the bridge and one for rescue. Measuring the time it takes to get the bridge in place and how long it takes to reach and save the targets and bring them back will be the determinant for how efficient the robot is. Low times for each would be ideal. The simplicity will be measured by the time, as if it is not simple, the time will be much higher than desired. The number of things that were 3-D printed can impact the time allowed for refining ideas and taking data. Making it over each obstacle on the first try is the ultimate goal.

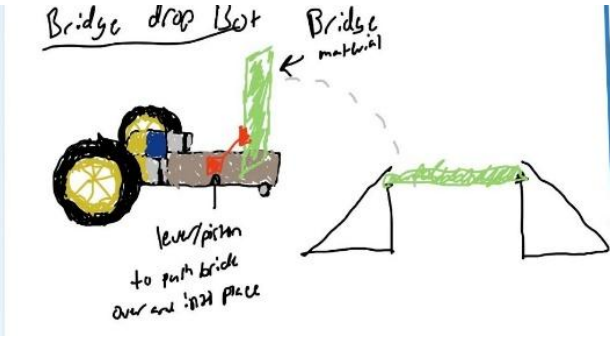
IDEATION

Concept Generation

In the ideation process, each team member sketched a small piece of what they pictured for a solution, and then passed the papers around and added to the other drawings and ideas. The bisociation technique was also used, which involved grabbing random objects and writing down the first things that came to mind in terms of descriptions and/or uses. After the ideation process was complete and each member had completed design concepts, the designs were input below.

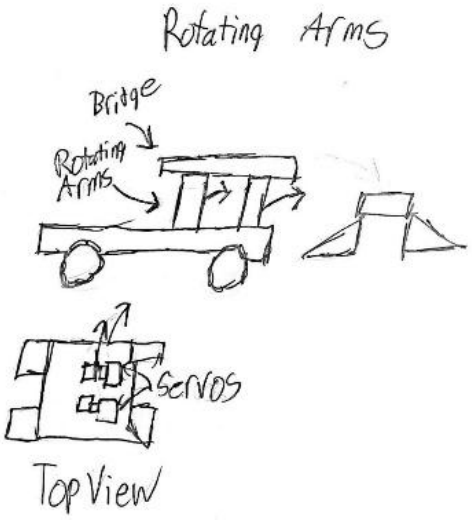
Generated Concepts

Figure #3: Bridge Drop Bot



This design employs a lever/piston to push the bridge off the bot and onto the ramp

Figure #4: Rotating Arms Bridge Bot



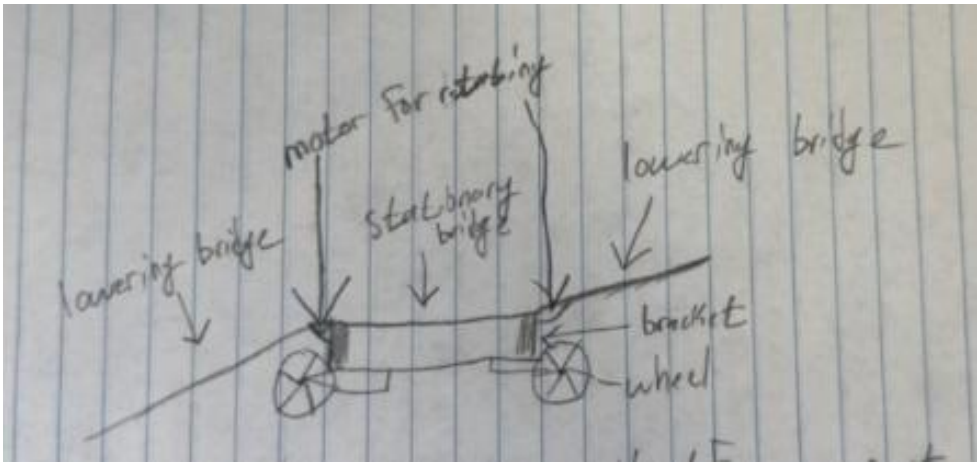
This design is a Bridger bot that holds the bridge above its head until it reaches the gap. Once it reaches the gap, the arms holding it up would rotate, allowing the bridge to fall between the gap. Servos would be used to accomplish this.

Figure #5: Claw Bridge

*Co-Pilot with Chat-GPT (2025 September, 22) "Create an image of a bridge robot that can rescue Lego people from lava across various terrain. This terrain includes a speed bump, a gap with lava below and a small wall. The Lego people are standing on the wall and waiting rescue. The robot(s) will be remote controlled and need to be able to navigate across this various terrain." Generated using Co-pilot with Chat GPT-5.

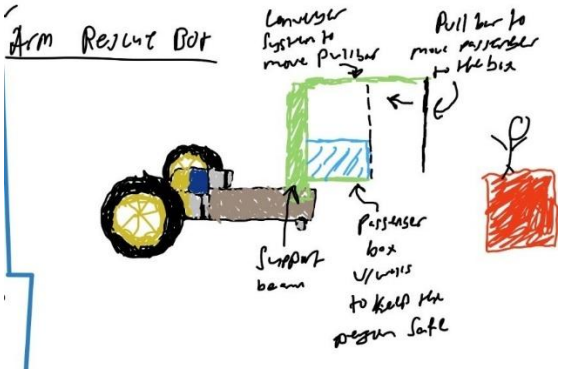
This design extends a bridge-like structure to the wall. The AI assumed the people were going to walk on it themselves. If the bridge were wider and had another from the back then it would've been more practical.

Figure #6: Mega Bridge



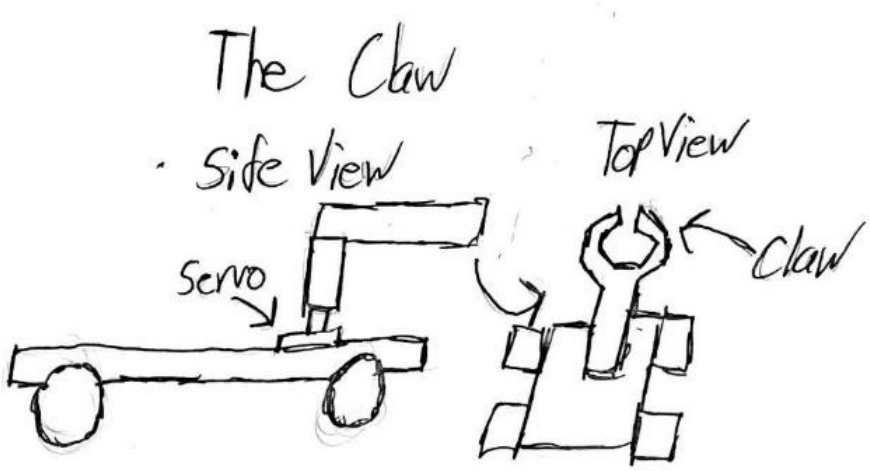
This robot uses itself as part of the bridge. As the bot approaches the gap, it lowers a bridge on both the front and back. This allows the rescue bot to drive right over it. Not Legal.

Figure #7: Box Rescue Bot



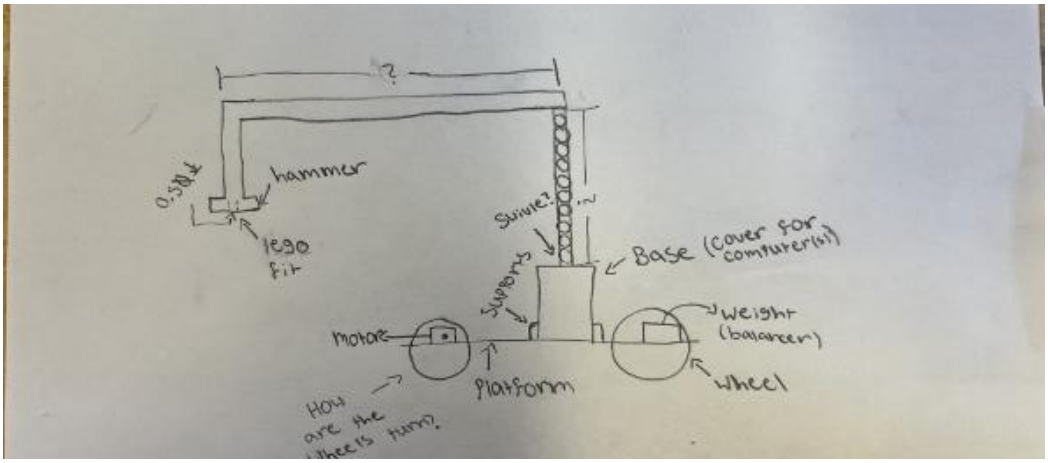
This robot uses a bar attached to a conveyor belt system to pull the people from the brick into a box. The support beam allows the robot to get close enough to the brick to be able to get the bar behind the people. The bar will also serve as one of the walls of the box to keep the people safe on their journey back across.

Figure #8: Claw Rescue Bot



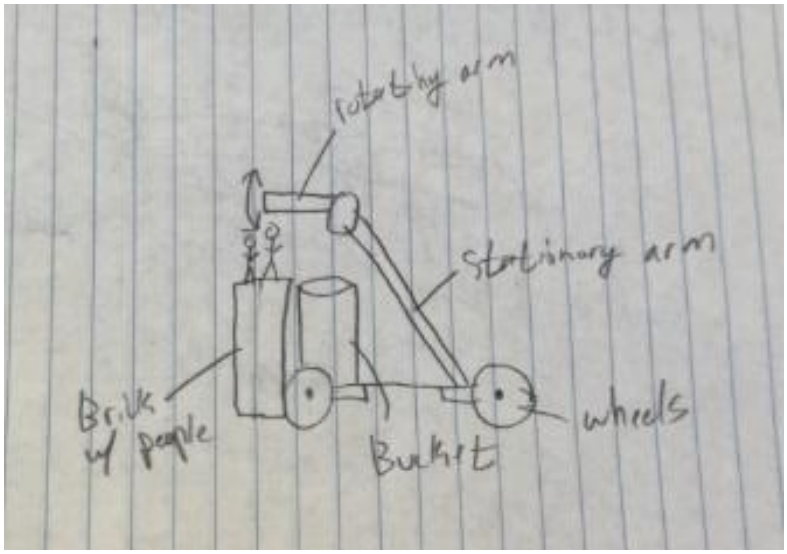
A rescue bot that would use a claw to grab onto the LEGO people. The claw would be able to rotate as well for ease of use. The claw itself would also be powered.

Figure #9: Crane Head Bot



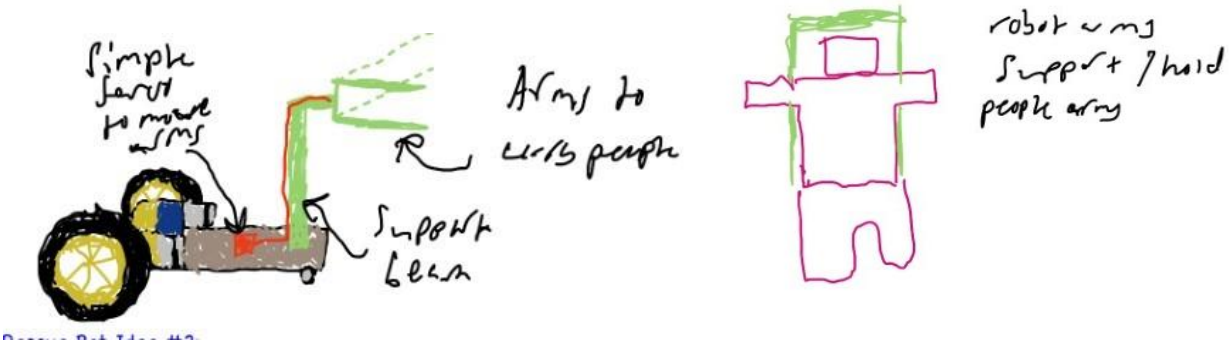
This robot would use two wheels only. It uses a crane-like arm to reach over and grab (or knock over) the Lego man. Uses weight to balance the robot, and an extension to extend the length of the arm.

Figure #10: Bucket Rescue Bot



This bot approaches the brick, leaving the bucket flush with the brick. The bot then uses a rotating arm to push the people into the bucket, keeping them securely in place.

Figure #11: Arm Bot



This robot uses a two-pronged crane that would be slightly bigger than the width of the people. It would be able to carry multiple people after picking them up from the brick. A servo would be employed to raise the arms up to ensure a safe return.

Concept Selection

Selection Criteria

The selection criteria are what the team felt were the most important factors in terms of building the robots.

Bridge Bot

- Ease of use – How easy it is to use.
- Time to get the bridge in place – How quickly can the bridge be placed.
- Maneuverability – How easy it is to move it around the obstacles.

Rescue Bot

- Ease of use – How easy it is to use.
- Maneuverability – How easy it is to move it around the obstacles and over the bridge.
- How many people can it carry – How many people can it save and transport at one time.
- Accuracy – How long does it take to get the bot in place to collect the people.
- Time to secure people – How long does it take to secure the people to return back to the start.

ResQTech

Screening Matrix

Table #1: Bridge Bot Screening Matrix

Selection Criteria	Bridge Drop	Rotating Bridge	Crane Bot	Mega Bridge	Ref
Ease of Manufacturing	+	0	-	+	0
Ease of use	+	+	+	+	0
Simplicity	0	0	-	0	0
Time to get bridge in place	+	+	+	+	0
Maneuverability	-	0	-	0	0
Pluses					
Sames					
Minuses	3	2	2	3	
Net	1	3	0	2	
Rank	1	0	3	0	
Continue?	2	2	-1	3	
	2	2	4	1	
	Probably not	Probably not	No	If Legal, yes	

This screening matrix for the bridge bot is quite segmented between ideas. Originally, the Mega Bridge was most desirable, but it was not legal for the competition, so that idea was

nullified. This left the Bridge Drop and Rotating Bridge. So far, ideas from both have been used in the prototype. The prototype resembles the Bridge drop the most thus far. It seems there will continue to be more refinement due to obstacles with placing the bridge.

Table #2: Rescue Bot Screening Matrix

Selection Criteria	Box Bot	Claw Rescue	Crane Bot	Bucket Bot	Arm Bot	Ref
Ease of Manufacturing	-	-	-	-	0	0
Ease of use	0	-	-	0	0	0
How many people it can carry	0	-	-	0	+	0
Accuracy	+	+	-	+	+	0
Simplicity	0	-	0	0	0	0
Time to secure people	0	-	-	0	0	0
Maneuverability	0	0	0	0	0	0
Pluses	1	1	0	1	2	
Sames	5	1	2	4	5	
Minuses	1	5	5	1	0	
Net	0	-4	-5	0	2	
Rank	2	4	5	2	1	
Continue?	Maybe	No	No	Maybe	Yes	

The decision was made to proceed with the arm bot, as it was considered the best option among all the ideas generated. Currently, issues are being encountered related to securing stranded individuals and bears. Efforts are ongoing to address and redesign the bot to meet specific needs. If the current approach becomes unfeasible, a design similar to the box bot or bucket bot will likely be pursued.

PROTOTYPE

Rescue Bot

Originally, the project began with the construction of a prototype for the arm mechanism using cardboard and popsicle sticks (Figure #12). Next, the design progressed to 3-D printing and laser cutting, resulting in multiple variations, including adjustments to the thickness of certain parts and changes to the gap between the arms (Figures #13-15). For the latest design, an arm was printed that rotates on a servo to scoop people and bears into a laser-cut box. (

Bridge Bot

The first prototype for the bridge bot consisted of a cardboard bridge with two popsicle stick arms attached to servo motors at the front. Subsequently, the front wheel portion of the bot was changed to a 3D printed cylinder, resembling a road roller. The bridge prototype was made of cardboard, featuring ramps and pillars at the bottom, serving as a reference for the appearance of the actual bridge. The semi-final design of the bridge includes two slots in the ramps, allowing the bot's arms to slide through and lift or draw the bridge. The arms, originally intended to be connected to the servos, require redesign to become smaller.

Figure #12: Initial Prototype of Arms for Rescue Bot

This is the initial prototype for the arms of the Rescue bot held together extremely loosely. It was designed to pick up the Legoman.

Figure #13: Angled View of Initial Rescue Bot Prototype

Above is the first rendition of the servo with the tower. The tower piece remained the same throughout this process, but the arm and servo structures had to be redesigned.

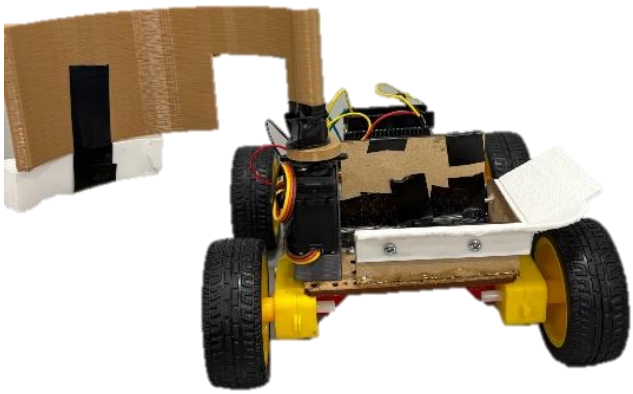
Figure #14: Front View of Initial Rescue Bot Prototype

As seen here, the tower and servo are not level, and in testing, the arm was not wide enough to reliably pick up the legoman.

Figure #15: Final Version of Failed Prototype

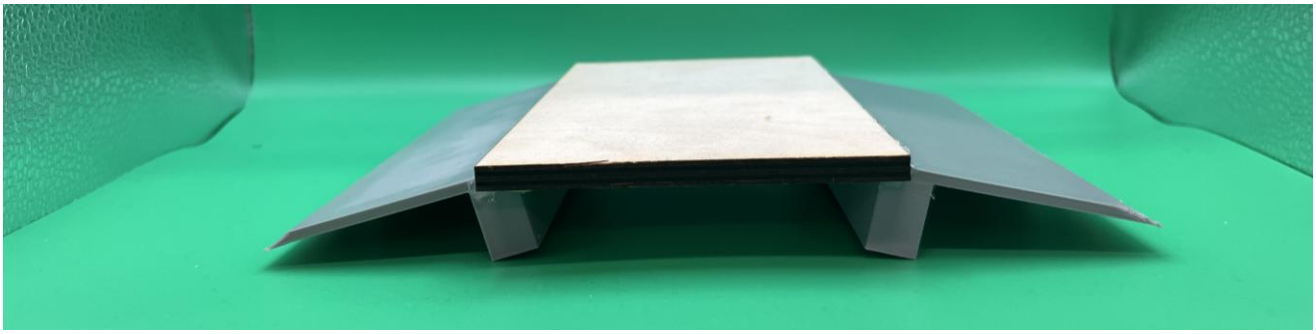
Seen above is the final scrapped rescue bot. As seen, the Arm is redesigned to be wider, and the servo has a 3d printed base. Ultimately, testing proved this prototype should be scrapped.

Figure #16: New Rescue Bot



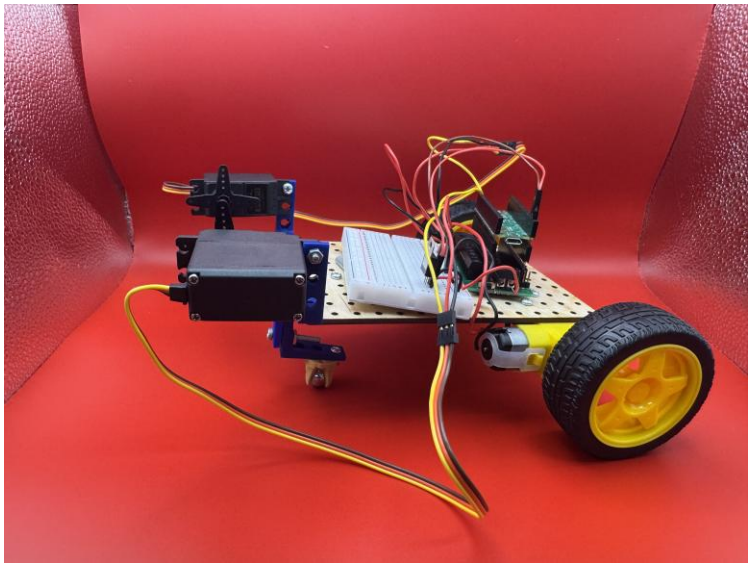
Seen above is the final rendition of the rescue bot, which employs a sweeper arm on a servo to sweep the people into the rescue basket.

Figure #17: Bridge



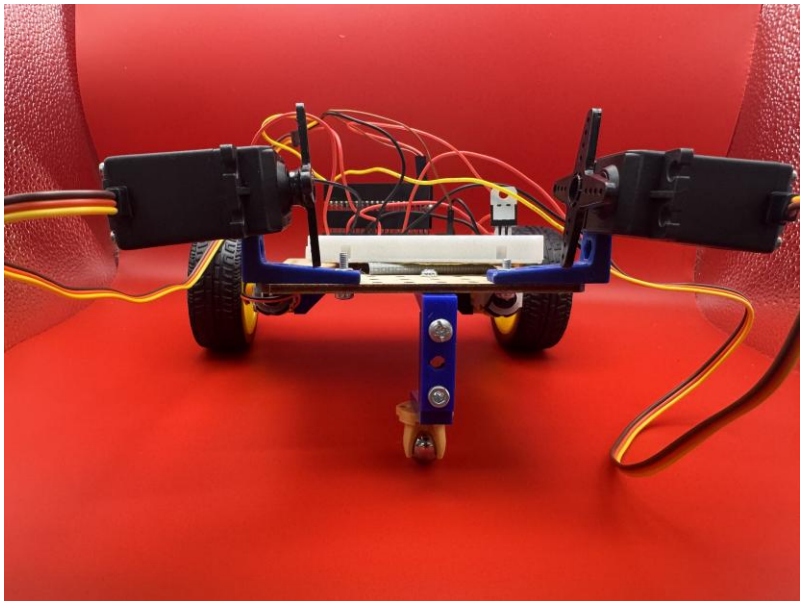
Above is the physical bridge to be placed to cross the bumps. It uses both laser-cut wood and 3d printed parts.

Figure #18: Initial Bridge Bot Design



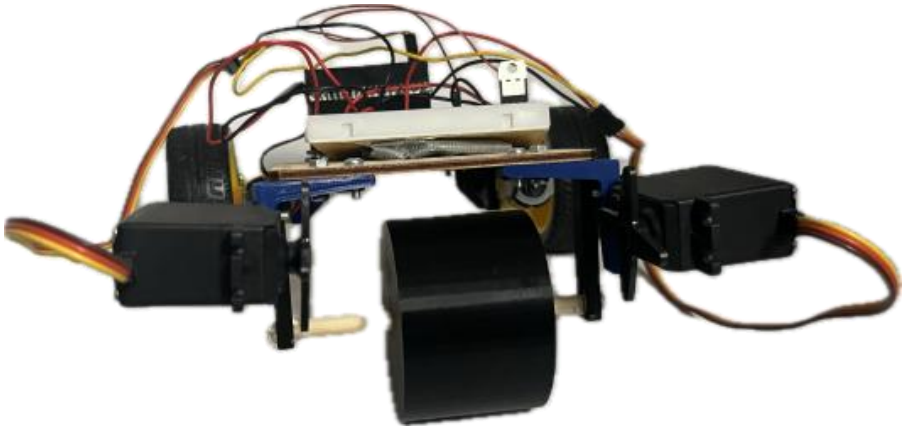
Above is the side view of the initial Bridge Bot design, using two motor wheels and a extended wheel bearing.

Figure #19: Front View of Initial Bridge Bot Design



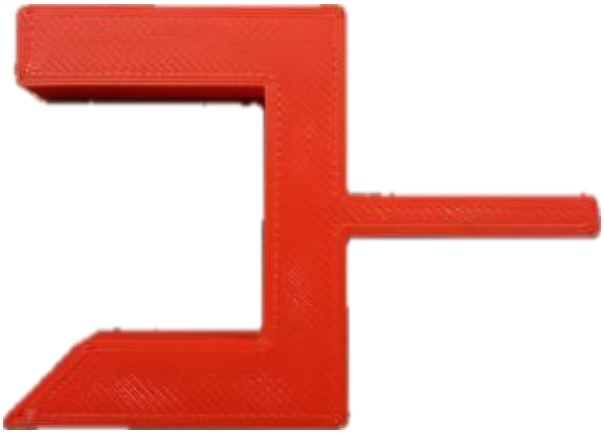
This front view shows the wheel bearing better, along with the two servos that will be used to deploy the bridge. This design struggled to get over the initial bump on the course.

Figure #20: Newer Rendition Bridge bot



Above is the second edition of the bridge bot, using a much larger wheel to surpass the bump on the course. It also lowered the servo position.

Figure #21: Bridge Attachment Part



Above is a model of the connection between the bridge and the bot itself to be used when deploying the bridge.

Figure #22: Final Design Project 2 Prototype

Above is the final DP2 prototype. It uses three IR sensors and a bucket sitting on top of the chassis to accomplish its goal.

TEST

Rescue Bot

The first test of the rescue bot included a few key parts. First, the ability to get over the initial bump was tested. Then, the ability to traverse the bridge consistently was tested. Finally, the ability to rescue the people and return them safely was tested. (Figure #22)

Bridge Bot

The first issue with the bridge bot was getting over the bump. With the road roller-inspired front wheel, the bot was able to get over the bump. As far as lowering the bridge, there have been difficulties lifting and drawing the bridge over the gap because of the design flaw from the previous arms; they were too big. After the design changes, the bot was able to successfully place the bridge in a secure spot for the rescue bot to traverse. (Figure #22)

Design Project 2

In its early stages, the autonomous bot faced difficulties navigating turns and often got stuck. This issue was resolved by modifying the code to better align with the course, resulting in consistent performance. However, the bot continued to struggle with determining the correct direction to take. Eventually all issues were resolved.

Figure #23: Test Data

Bot Type	Criteria		Scores
Bridge Bot	Clears Speed Bump (1st)	5	5
	Deploys Bridge	20	20
	Places Bridge on Notch	20	20
	Clears Speed Bump (2nd)	5	5
Rescue Bot	Clears Speed Bump (1st)	5	5
	Traverses Notch	10	10
	Retrieves Stranded Indv.	20	20
	Clears Speed Bump (2nd)	5	5
	Brings Indv. Back to Start	10	10
	Elapsed Time		1:04

CONCLUSION

The redesigned Rescue bot has shown considerable improvements, outperforming the original prototype. Comprehensive testing was successfully conducted during practice sessions and two rounds on the day of the Bot Bash. The addition of an arm and holder proved to be highly effective in addressing the challenges faced. Throughout the testing process, no further issues were identified. Similarly, the bridge bot completed all tasks successfully at the Bot Bash, with the refined bridge design contributing significantly to its overall success at the competition. Regarding Design Project 2, there was also notable success. Due to time constraints leading up to the Bot Bash, there wasn't much opportunity for iteration, which resulted in some minor issues. Nonetheless, the bot efficiently navigated the maze in approximately 40 seconds during its best run. Ultimately, the team secured an impressive 5th place overall in points and achieved 1st place in the presentation category.

Appendix

CODE

The code for both the Rescue Bot and Bridge Bot in Design Project 1 is relatively straightforward. Both robots utilize a preconfigured WNE103 Python script that leverages the Pico board, Kitronik motor shield, and the on-board Wi-Fi module of the Pico, along with the Sunfounder app for remote control via a smartphone. A provided script manages the interaction between the Pico board and the remote-control app. The programming involves selecting the functions of the app's buttons and coding the activation of corresponding servos and wheels. Below is the main Python script that governs the bots.

```python
import WNE103
import json
import time
import machine

time.sleep(3)
#  creates the ap for the phone to connect to
ws = WNE103.Controller_Server("team155rescue","blahblah") # init websocket

#  initializes the motor shield and the led
led = machine.Pin("LED", machine.Pin.OUT)
led.off()
board = WNE103.KitronikPicoMotor()
S1=WNE103.Servo(15)

#  function that handles if the buttons are pressed
def handle_request(data_sent, ws):

    #  forwards, left, right, backward, stop, may not be in dict
    try:
        dpad_state = data_sent['K']
    except KeyError:
        dpad_state = False

    #  either true or false, may not be in dict
    try:
        button1_state = data_sent['Q']
    except KeyError:
        button1_state = False

    try:
        button2_state = data_sent['R']
    except KeyError:
        button2_state = False

    try:
        button3_state = data_sent['S']
    except KeyError:
        button3_state = False

    try:
        button4_state = data_sent['T']
    except KeyError:
        button4_state = False


    #  Handles the motion side
    if dpad_state:

        #  figures out which way the robot wants to move
        if dpad_state == 'forward':
            led.on()
            board.motorOn(1, 'f', 100)
            board.motorOn(2, 'f', 100)

            ws.send_dict['A'] = 100
            ws.send_dict['D'] = 100

        elif dpad_state == 'left':
            led.on()
            board.motorOn(1, 'r', 80)
            board.motorOn(2, 'f', 80)

            ws.send_dict['A'] = -100
```


Here on this screenshot, majority of the code listed is the same in both bots. The only difference found will be the Servos directly to the left of this text.

The only difference between the Rescue Bot and Bridge Bot is that the Rescue Bot has one servo, as seen in the code here, while the Bridge bot has two.

```
            ws.send_dict['D'] = 100

        elif dpad_state == 'right':
            led.on()
            board.motorOn(1, 'f', 80)
            board.motorOn(2, 'r', 80)

            ws.send_dict['A'] = 100
            ws.send_dict['D'] = -100

        elif dpad_state == 'backward':
            led.on()
            board.motorOn(1, 'r', 100)
            board.motorOn(2, 'r', 100)

            ws.send_dict['A'] = -100
            ws.send_dict['D'] = -100

        else:
            led.off()
            board.motorOff(1)
            board.motorOff(2)

            ws.send_dict['A'] = 0
            ws.send_dict['D'] = 0

    #  Handles the button pressed
    if button1_state:
        S1.STServo(180)

    elif button2_state:
        S1.STServo(150)

    elif button3_state:
        S1.STServo(90)

    elif button4_state:
        S1.STServo(0)

    ws.write()


def main():
    ws.start()
    print("start")
    while True:

        status,result=ws.transfer()

        if status == True:
            print(result)
            handle_request(result, ws)

        time.sleep_ms(100)


try:
    main()
finally:
    ws.stop()
    print('stopped')
```

Once again majority of the code is identical, except for the if and elif statements with the buttons. To the left is the code found in the Rescue bot where each button directs the servo to a different angle. This is what allows the arm to swing.

The Bridge bot takes a very similar approach, with different angles to drop the bridge.

For Design Project #2, the code was created from scratch. It is not necessarily more complicated; it simply has more components. In this instance, two motors were utilized, once again managed by a preprogrammed script. The only external parts used were three Sharp IR Sensors, which helped navigate the course. Using readings from the IR sensors, the bot determined which direction to slowly turn. Additionally, multiple failsafes were included to prevent the bot from getting too close to the wall, and a turn counter was implemented to avoid the bot becoming stuck.

ResQTech

12/9/25, 9:52 PM [main.py]

```
from machine import ADC, Pin
import WNE103
import time
from time import sleep
from machine import Pin
# Call an object for the Kitronik shield
board = WNE103.KitronikPicoMotor()

speed = 80#motor speed


Front = ADC(26)#GP26_A0
Left = ADC(27)#GP27_A0
Right = ADC(28)#GP28_A0

TurnLefts = 0
TurnRights = 0

def Reverse():
    board.motorOn(1,'f',speed)
    board.motorOn(2,'f',speed)#forward is reverse for motors
def RightT():
    board.motorOn(1,'r',speed)
    board.motorOn(2,'f',speed)
    sleep(0.01)
def LeftT():
    board.motorOn(1,'f',speed)
    board.motorOn(2,'r',speed)
    sleep(0.01)
def Forward():
    board.motorOn(1,'r',speed)#reverse is forward
    board.motorOn(2,'r',speed)
    sleep(0.05)
while True:
    sleep(0.005)#how quick to reoccur the function

    Frontdist = Front.read_u16()#value in bits
    Leftdist = Left.read_u16()#value in bits
    Rightdist = Right.read_u16()#value in bits, refrence chart from original sensor data
    board.motorOn(1,'r',speed)#go forward while idling
    board.motorOn(2,'r',speed)#go forward while idling, reverse is forward
    print(Frontdist,"Forward")
    print(Leftdist,"Left")#TESTING PURposes
    print(Rightdist,"Right")

    #fail safe first
    if Frontdist >= 61000: #front facing failsafe, about 2 inches
        Reverse()
        sleep(0.5)
    if Frontdist <= 15000 and Leftdist <= 15000 and Rightdist <= 15000: #don't fall off cliff
        board.motorOff(1)
        board.motorOff(2)
        sleep(15)
    if Leftdist < Rightdist: #important to remember less is further, high is closer
        LeftT()#may have to change/flip
        TurnLefts = TurnLefts + 1
        TurnRights = TurnRights - 1
        #print("Turning Left")
        sleep(0.04)
        Forward()
        sleep(0.05)
    if Rightdist < Leftdist:
        RightT()
        TurnRights = TurnRights + 1
```

As seen before there are the usual imports.

First are calls to the IR sensors, the speed set for the motors, and the turn counter.

Next are the motions defined for the motors

This is where the constant function begins. It starts with defining IR sensor distances, and includes testing prints

Next begin some of the failsafe's and the stop function

Overall logic of turning

12/9/25, 9:52 PM [main.py]

```
65          TurnLefts = TurnLefts - 1
66          #print(TurnRights)
67          #print("Turning RIght")
68          sleep(0.04)
69          Forward()
70          sleep(0.05)
71      if Rightdist >= 61000:
72          Reverse()
73          sleep(0.3)
74          LeftT()
75          TurnLefts = TurnLefts + 1
76          TurnRights = TurnRights - 1
77          sleep(0.13)
78          #board.motorOff(1)
79          #board.motorOff(2)
80          sleep(0.01)
81      if Leftdist >= 61000:
82          Reverse()
83          sleep(0.2)
84          RightT()
85          TurnRights = TurnRights + 1
86          TurnLefts = TurnLefts - 1
87          sleep(0.13)
88          #board.motorOff(1)
89          #board.motorOff(2)
90          sleep(0.01)
91      if Rightdist < 12000:
92          Reverse()
93          sleep(0.1)
94          LeftT()
95          TurnLefts = TurnLefts + 1
96          TurnRights = TurnRights - 1
97          sleep(0.04)
98          #board.motorOff(1)
99          #board.motorOff(2)
100         sleep(0.01)
101     if Leftdist <12000:
102         Reverse()
103         sleep(0.1)
104         LeftT()
105         TurnLefts = TurnLefts + 1
106         TurnRights = TurnRights - 1
107         sleep(0.06)
108     if Frontdist > 58000 and Leftdist >50000:
109         RightT()
110         TurnRights = TurnRights + 1
111         sleep(0.09)
112         #board.motorOff(1)
113         #board.motorOff(2)
114         sleep(0.01)
115     if TurnRights > 30:
116         TurnRights = 0
117         sleep(0.001)
118         Reverse()
119         sleep(0.2)
120         LeftT()
121         sleep(0.11)
122         Forward()
123         sleep(0.1)
124     if TurnLefts > 30:
125         TurnLefts = 0
126         sleep(0.001)
127         Reverse()
128         sleep(0.2)
129         RightT()
```

More failsafe's to not get too close to wall

The following if statements were determined useful in testing where the bot often struggled

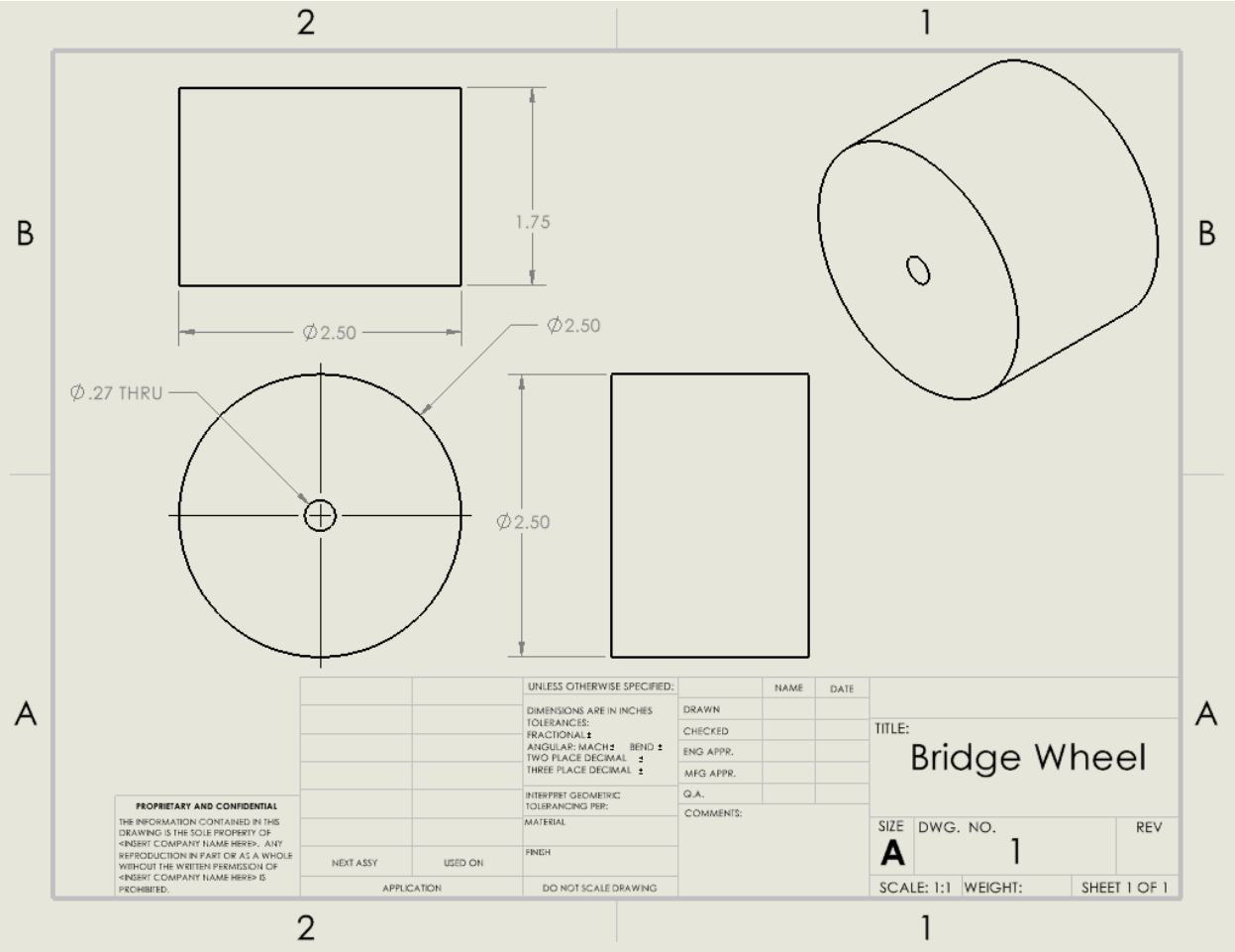
If IR sensors are wrong

Last is the turning counter. This helps determine when the Bot may be stuck, acting as a sort of memory

It is noted that there could be greater refinement in the code listed above. More precise data and a greater use of a 'memory' would be helpful. Still, the code is quite successful.

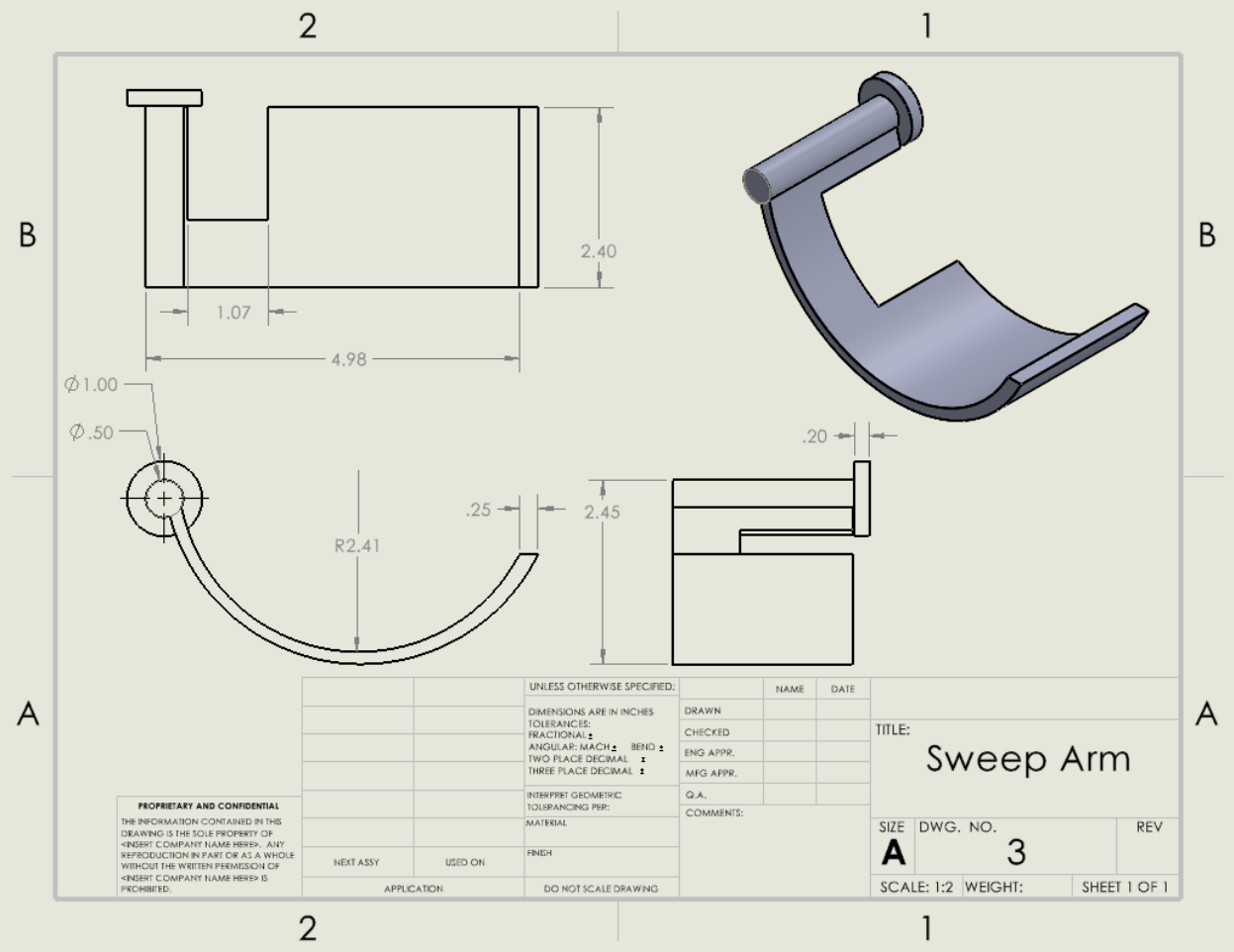
DRAWINGS

First is the main aspect of the Bridger bot the wheel:

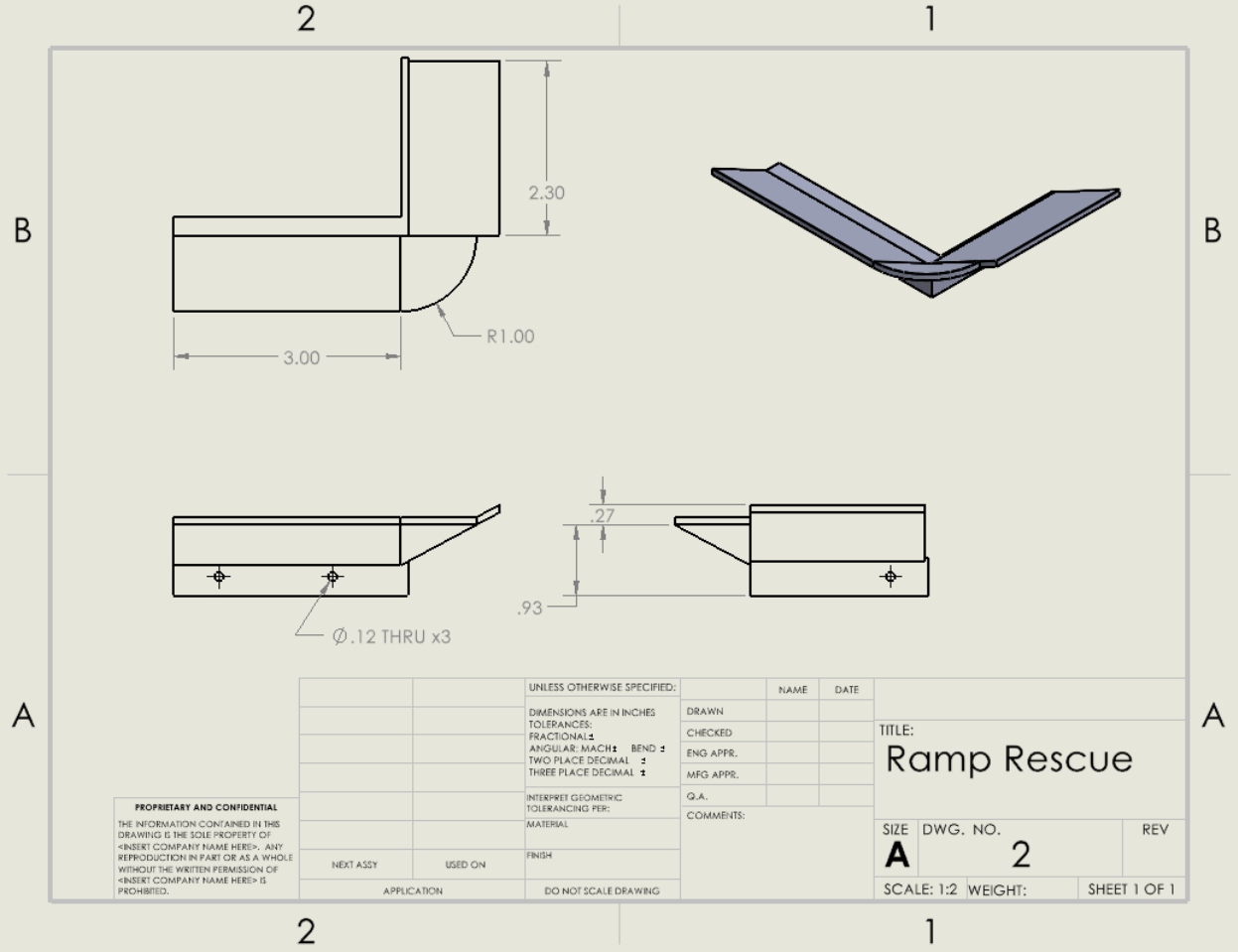


Drawing 1: Above is the wheel used on the Bridge Bot

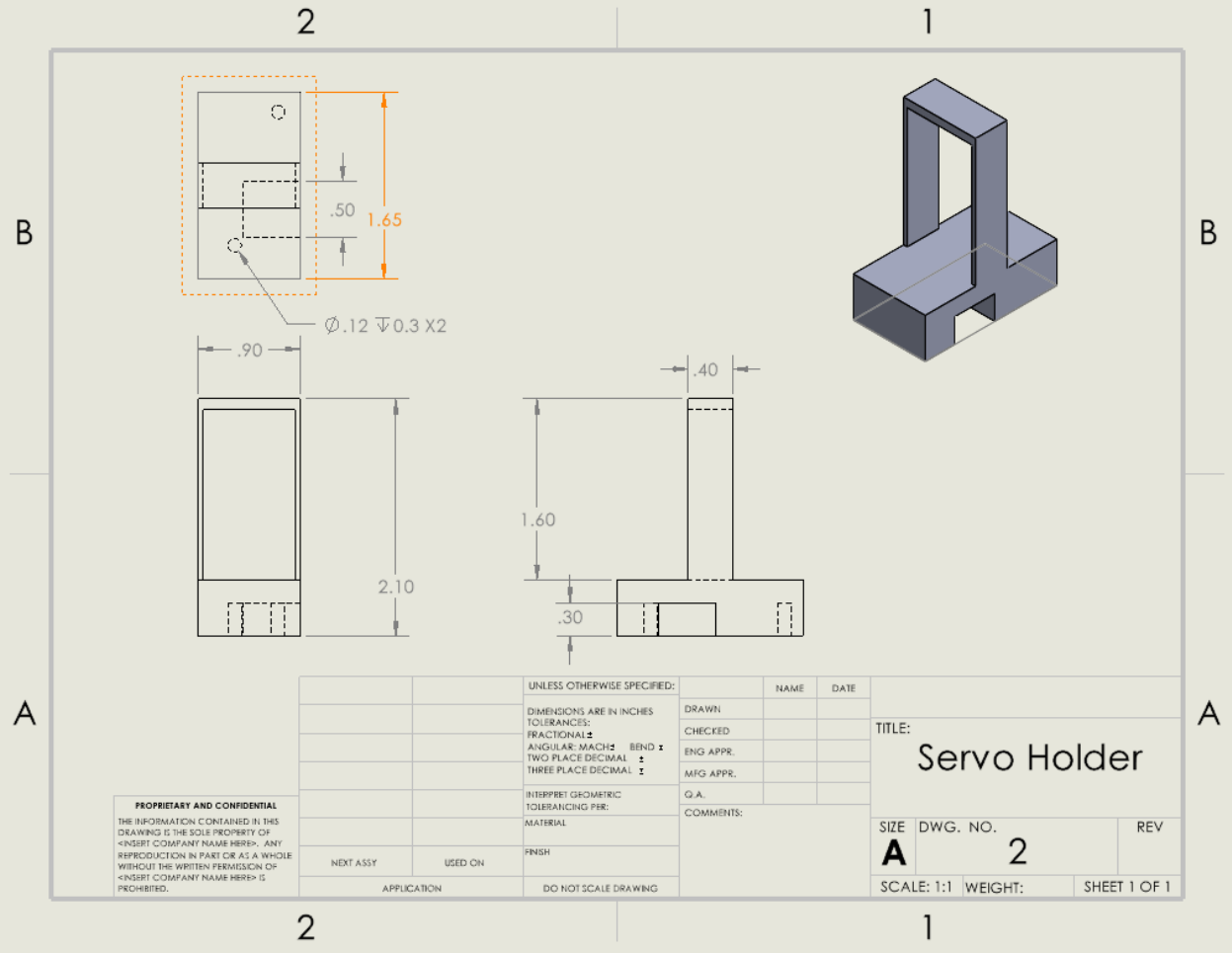
Moving onwards to the Rescue bot there are a few parts of note.



Drawing 2: Sweep Arm Drawing



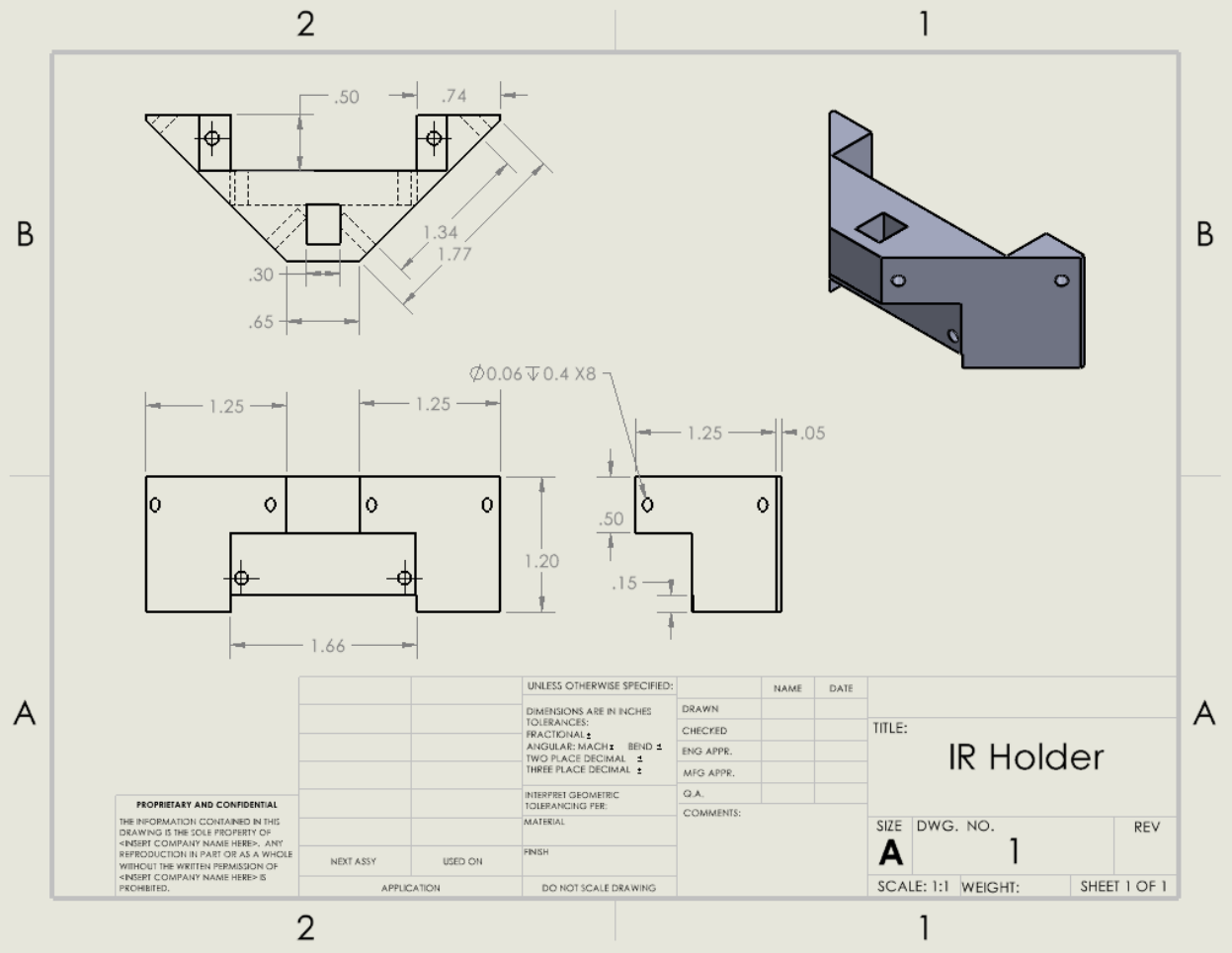
Drawing 3: Rescue Ramp



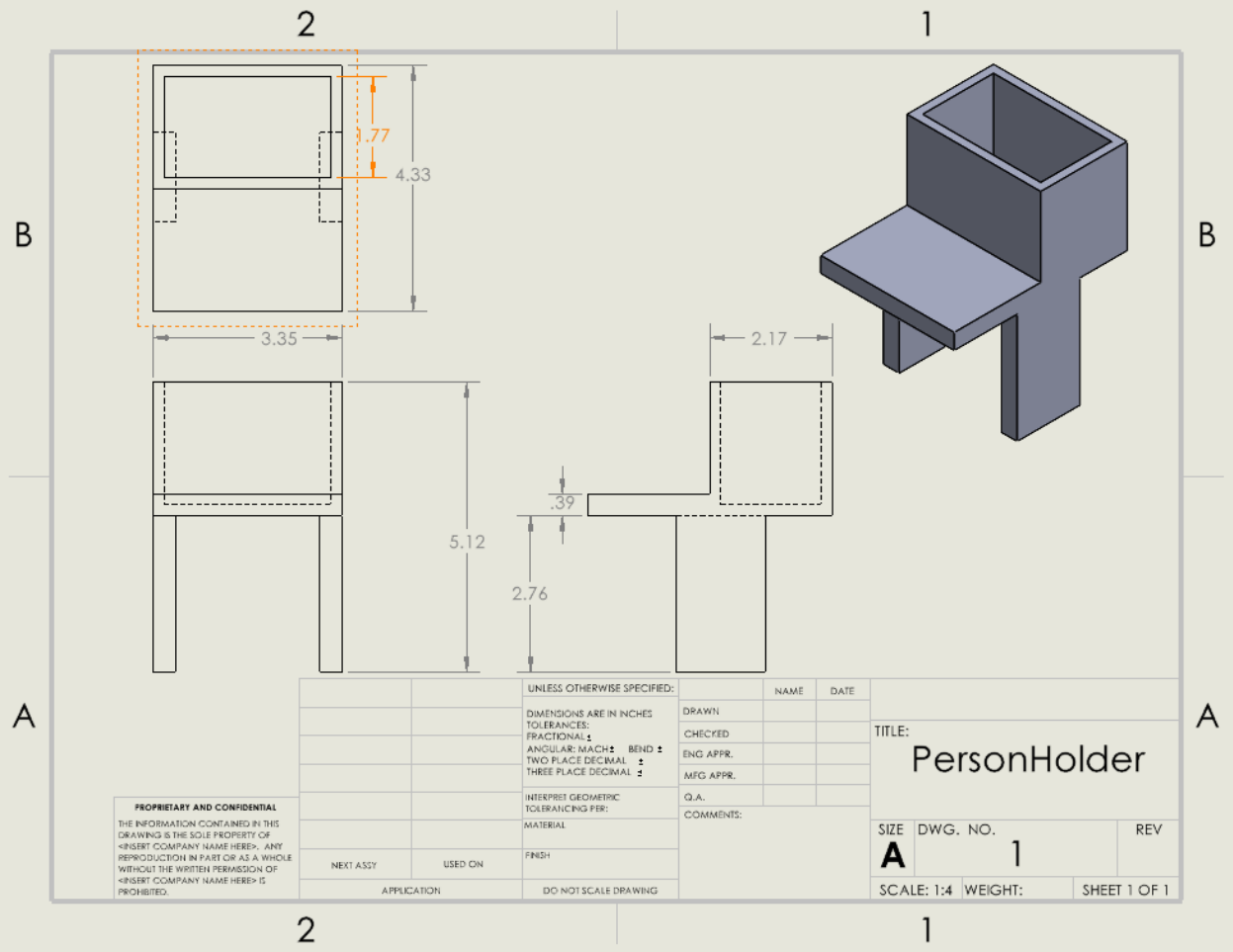
Drawing 4: Servo Holder

The remaining pieces of wood were 1/8-inch fiberboard, all laser-cut and rectangular, for the Rescue Bot. Every other part was already provided, with little to not changes made.

There are two pieces to be noted for Design Project number 2.



Drawing 5: IR Holder



Drawing 6: Person Holder

Above were all manufactured parts, not including laser-cut wood pieces.