KIRBY-TE

Design Report

Caden McCarthy
cadenm@uci.edu

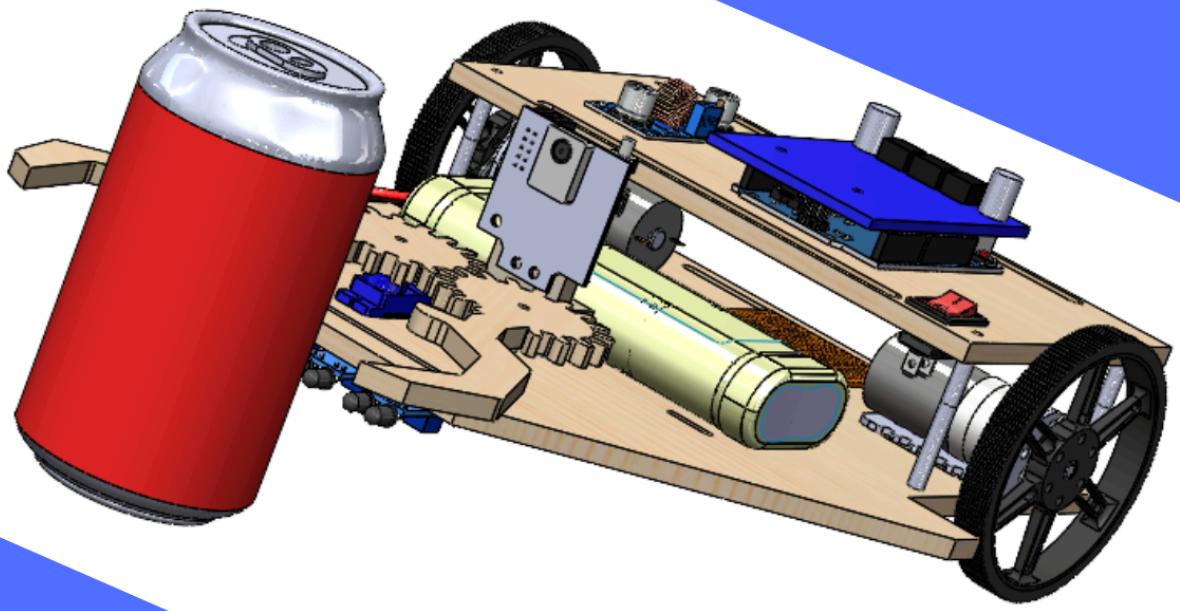
Angel Derouin
ariverai@uci.edu

Eric Cheng
chenge17@uci.edu

Gavin Nguyen
gavinn2@uci.edu

Kristen Chung
chungks1@uci.edu

Victor Plesco
plescov@uci.edu



Winter 2023
University of California, Irvine
ENGR 7B: Introduction to Engineering

Table of Contents

Executive Summary 2

Problem Definition 3

- Introduction 3
- Technical Review / Background 3
- Design Requirements 4

Design Description 5

- Summary of Design 5
- Design Details 6
- Wiring Diagram 9
- Algorithm Design 9

Action Item Report 18

- Task Assignment 18
- Gantt Chart 18

Evaluation 19

- Calculations 19
- Test Plan 20
- Results & Discussion 23

Appendix A: SOLIDWORKS Drawings 25

Appendix B: Bill of Materials 29

Appendix C: Arduino Autonomous Mapping Code 30

Appendix D: References 37

Executive Summary

Our team was tasked to create and design an autonomous rover that can line follow and retrieve a can at the end of a closed track. Our design parameters involved designing a claw that could grasp the can, utilizing infrared sensors for line following, coding a PixyCam for object detection, and fabricating a compact chassis that could maneuver around the track and hold all of the components together. Our rover also needed to fully stop at the end of the track, then begin object detection to retrieve the can. Our main goal was to create a compact rover that could swiftly maneuver throughout the course, and successfully detect the can for retrieval.

One unique aspect of our rover is the formation of our IR sensors. The IR sensors are placed below the chassis in a triangular formation. This allows us to be more accurate in detecting the black line for our line following code. Our chassis is also shaped like an arrow, and features slits to strap down our battery and holes for standoffs. On top of our chassis, we created a platform supported by standoffs to hold the Arduino board. Our claw mechanism utilizes one 3D printed claw and one laser cut claw. They are connected through gears which allow the claws to open and close due to the rotation of the micro servo. For our PixyCam, we created a holder at the front of the chassis that allows the camera to have full visibility of the environment.

We decided on the 34:1 gearmotor as it had a higher maximum efficiency (44%) and a 300 RPM no-load speed compared to the 47:1 gearmotor, which had a 220 RPM no-load speed and a maximum efficiency of 42% (both at 12V). We implemented the 80mm wheels into our rover as it gave us greater ground clearance for the sensors. Our chassis was laser cut to reduce cost and also because it was easier to secure our electronics with the slits.

Fabrication of our entire rover assembly was complete by Week 7. In the remaining weeks, we focused on testing the line following and can retrieval code. Through testing, we discovered some issues with our line following. We determined that these issues were due to the bending of our IR sensors that caused the internal components to break. To solve this problem, we soldered the broken parts back together. Another issue we encountered was the turn speed of our rover. By testing different motor speeds and removing delays from the code, we were able to shorten our track time by ten seconds. We also adjusted the IR sensor distances by centering them according to the midpoint of the two bulbs of the IR sensor. This was different from our original placement, which had the black receiver bulb over the black line. Lastly, we found that our claw mechanism was slightly loose due to the screw connected to the servo motor being of nonoptimal length. By adding washers we were able to improve the stability of our claw for can grabbing.

For our final track results, we were able to complete the entire course in 20.5 seconds. Our rover was significantly faster during turns from our previous trials. One thing we tried to improve upon was our consistency for the line following code. During some tests, our rover could not detect the line and ran off the course. However, through trial and error, we were able to place second in the overall final competition.

Problem Definition

Introduction

Utilizing skills in coding, fabrication, electronics, and mechanical design, we underwent a 10 week process to create an autonomous rover within a budget of $300. The rover should be able to use a PixyCam for object detection, implement infrared sensors for line following, maneuver the preliminary and final tracks successfully, stop when encountering the end of track, and retrieve a can as efficiently as possible.

Our main priority was to create a rover that was light and compact to be able to easily make sharp turns around the track for line following. We also wanted to ensure that the claw mechanism was at the optimal strength in order to grab and retrieve the can. Keeping these constraints in mind, our goal was to determine the best components for our rover and how to implement them into our overall design.

Technical Review / Background

Robotics has been a field of interest for humans throughout history. For almost two centuries, humans have always been fascinated by the idea of controlling a small vehicle capable of being controlled by humans from far distances. To do this, Nikola Tesla, an innovator in electronics, decided to create some of the first vehicles controlled by a variety of radio remote controls. The addition of the radio controllers allowed the individual to control the vehicle from far distances, allowing exploration and research from places where people usually are not able to go.

This innovation of a now radio controlled vehicle was soon improved during the Industrial Revolution. As the Industrial Revolution continued, notable countries became industrialized, and the robotic and rover industries began flourishing due to the innovative technology surrounding it. By the year 1980, the market of robotics and rovers began to bloom due to competition between global superpowers, such as the United States and the USSR. The competition between the two countries soon became too big to be held on Earth and moved to outer space. The space race between the USSR and the United States helped fund the research and technology to make rovers.

Space exploration soon became one of the main priorities of the United States. A subteam of mechanical engineers, led by Eduardo San Juan, worked to develop and invent a Lunar Rover to soon be launched into space and used by astronauts on the moon. By 1978, the Moon Buggy was a well-known revolutionary advancement in not only space exploration but rover technology. To expand space exploration one step further, NASA launched a rover named “Pathfinder” to land on Mars and demonstrate just how revolutionary rover technology could influence space exploration on the Red Planet. Pathfinder was one of many autonomous rovers to be sent to Mars for exploration. Rovers have become one of, if not, the most important factors contributing to the development and exploration of planets and natural satellites.

In addition to the work of NASA, the space agencies of other countries, such as China and Russia, have further implemented autonomous rover technology in their planetary exploration

missions. The space environment makes it necessary for the rovers to be autonomous, just like our project, to be able to search for and send data back to Earth.

Our project is dedicated to design and fabricate an autonomous rover that is capable of line following and object retrieval. Similar to space exploration, our rover will be directed to follow a specific track to then collect the object that it is programmed to, allowing us to grasp onto a small replicated experience of the space exploration rover program. This autonomous project will then allow us to experiment with the engineering analysis and fabrication that had to be involved to effectively replicate a basic understanding of exploration along with path and object recognition. This project allowed us to test our problem-solving skills. In addition, a series of learned engineering skills had to be applied to design a sufficiently effective claw mechanism capable of picking up the object, along with a chassis that would facilitate turning and effectively distribute the weight of the individual rover components. The coding subteam was able to brainstorm, test and troubleshoot a series of algorithms to efficiently meet the line following and object recognition requirements. To conclude, the autonomous rover project allowed our team to put our engineering abilities to the test and to gain knowledge in programming and fabrication.

Design Requirements

1. Structure:
    a. Rover dimensions < 12 x 16 inches including wheels and claw
    b. Use materials provided by the lab; other materials can be purchased externally with permission from TA
    c. 3D printed and laser cut components are allowed but added onto the budget
    d. Battery must be easily removable
2. Power:
    a. Battery switch must be easily accessible
    b. Using an internal combustion engine is prohibited
3. Safety:
    a. No protruding sharp objects. All sharp corners must be filed or sanded down
    b. Wires and connectors are to be completely covered and insulated
    c. All batteries will be charged by staff members
4. Cost:
    a. Budget must be less than $250; incorporating 3-D printing and laser cutting will increase the budget to $300
    b. The cost must be broken down into a Bill of Materials (BOM)/Parts List, in which the Fair Market Value (FMV) of each component must be listed.
5. Product:
    a. The rover must follow a course denoted by a black line and grab a payload

Design Description

Summary of Design

The Kirby-te team's rover was designed and influenced by the design of the track that it was going to be tested on. Consequently, the overall design was meant to be small to avoid any issues regarding the turning radius of the rover as a whole. In addition, the various electrical components that made it to the final design of the rover altered the overall design so that they would all fit onto the chassis.

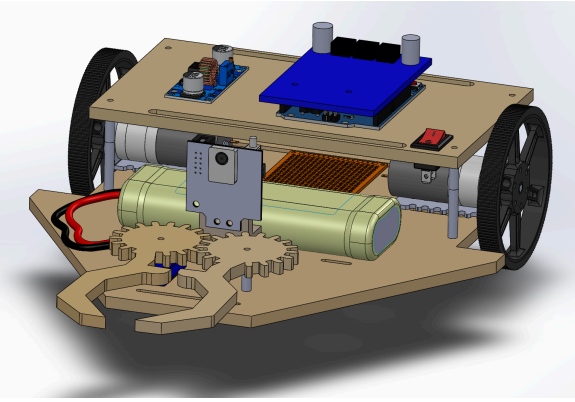
The claw mechanism was designed to tightly hold the circular object tightly. In order to do so, it had to apply a grabbing force on both sides of the object. To achieve this, the claw mechanism is controlled by a single servo on the right side of the front of the rover. The right 3D printed claw would be mounted by the servo horn and a single servo at the center to tightly secure the claw onto the servo to avoid any grip or control issues. Consequently, a gear mechanism was created to efficiently control both left and right claws from a single side. Each claw consisted of a gear with 18-teeth to avoid any chipping and improve the torque of both sides. The left wooden claw was mounted strategically on a short standoff to have equal height to that of the servo to have both gears meshing correctly and efficiently.

The wheels were chosen to be placed at the back of the rover. This design position was made due to the majority of the electronics and the battery being towards the back of the rover and thus requiring the largest amount of support. The single ¾" Caster Wheel was mounted along the vertical center line towards the front of the rover to accommodate any weight from the battery, the servo, or claws that may cause the rover to tip. Aside from distributing the weight evenly across the chassis, placing the wheels in a triangle formation was the best way to support the overall weight of the rover.

The chassis was originally designed to be a two-platformed circular chassis with vertical mounts on its edges. This idea was quickly scrapped due to its bulkiness. The overall final design of the chassis is a pentagon shape to reduce its size and to aid in the placement of the IR sensors at the bottom of the chassis. A series of slots and cutouts were made throughout the chassis for wire management purposes. These series of slots and cutouts not only improved the wire management but reduced the weight of the rover to make it lighter and faster. A second platform was created at the back to efficiently help locate and distribute the rover's electrical components and avoid any short circuits that may end up causing damage to the electrical systems. The slots in the middle of the chassis base were placed to wrap velcro straps around the battery to safety and tightly hold it in place. A series of M3 screws were placed across the chassis to hold the electrical components, along with standoffs and brackets. Laser cutting was effectively used for the creation of both the chassis and the back platform to avoid any dimension errors caused by other fabrication methods.

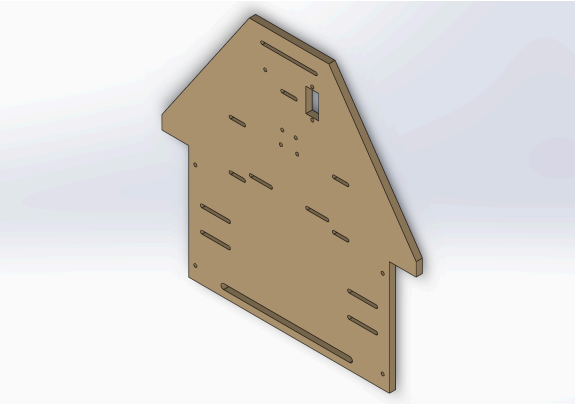
Design Details

Isometric View: Rover Full Assembly plus C.O.G.



This is an isometric view of the integrated rover assembly. The assembly was a useful way to view and test how well the electric components were spaced out, and how everything was distributed and would work together as a rover. The height of the rover is perfectly leveled due to the back two wheels and the caster wheel creating a balanced chassis. In addition, the claw mechanism is located at the front of the chassis. The PixyCam is located 2.51 inches behind the front to aid and provide the best field of view of the camera. The width of the total assembly of the rover is a total of 9.5 inches and a length of 12.75 inches in total from the back to the tip of the claws.

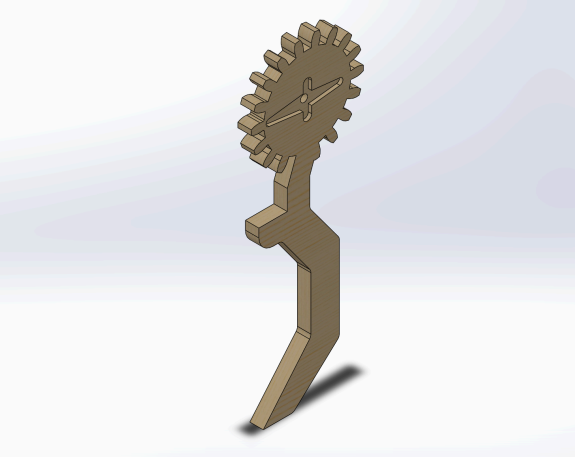
Isometric View: Chassis



The chassis was the most crucial part of the assembly. Its design allowed for an optimal distribution of weight across it. The chassis measures a total of 9.5 inches in length and 8 inches in width. Several M3 screw holes were located across the platform to aid the support of electronics, platforms, and mounting brackets. Various slots were created to aid both with wire management and the placement of the IR sensors. For the placement of the IR sensors, 3 mm diameter slots were

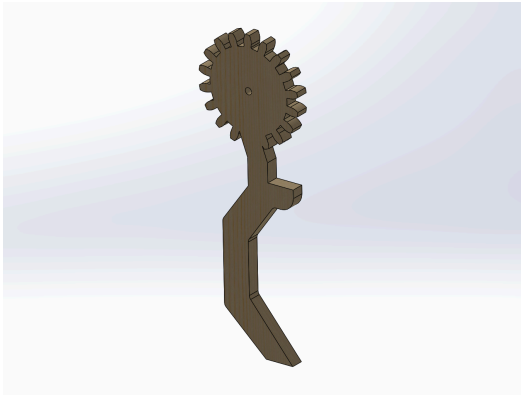
created instead of screw holes. This allowed the fabrication team to move the IR sensors in the event it was required to facilitate and ease the troubleshooting process with line following. A cutout was created for the micro servo to securely set it in place flush to the chassis. Slots were also created for the motor brackets to increase the adjustability of motor placement.

Isometric View: Right Claw



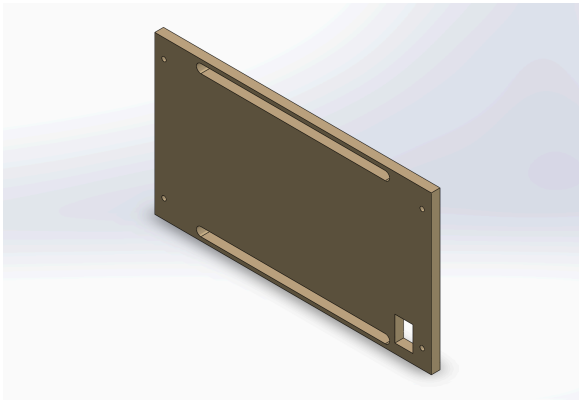
The right claw was a crucial part of the overall design. It allowed the micro servo to control both the right and left claw. The right claw was 3D printed to include the micro servo horn cutout, increasing its grip and securely fastening it to the micro servo. In addition, the claw is combined with the gear to prevent any strength issues that it would have had if this were not done. The overall length of the claw is 5.6 inches. The claw design was strategically designed to, when meshed with the left claw, have a grip diameter of 2.2 inches. This 2.2 inch grip length is 0.4 inches less than the average diameter of a soda can, allowing the claw mechanism to tightly and safely secure the can with no grip issues. The gear has a radius of 0.76 inches and a total of 18 teeth.

Isometric View: Left Claw



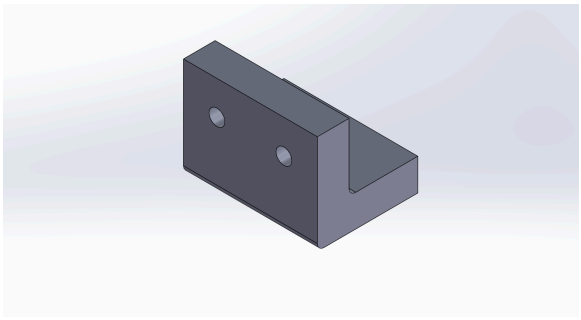
This is an isometric view of the left claw of our rover. The left claw measures a length of 5.6 inches. Similarly to the right claw, the left claw is merged to its gear to increase its durability. The hole at the center of the gear is an M3 screw hole to securely tighten it to a standoff on the chassis. The gear's radius is 0.76 inches, with a total of 18 teeth, like the right claw.

Isometric View: Top Platform



This is an isometric view of the back platform which held both the buck converter and the Arduino and its motor shield. This component was laser cut to avoid dimension errors and increase fabrication speed. There are a total of four screw holes, each placed 0.25 inches from their closest edge. These four screw holes were made to, with the help of an M3 screw, attach the platform to standoffs. Two long slots are placed at the front and the back of the platform, and were created to help with wiring management. In addition, a hole was created for the on/off switch. This cutout is 0.67 inches by 0.5 inches to tightly fit the switch. Overall, the back platform was crucial for both wire management and the equal distribution of electronics on the rover.

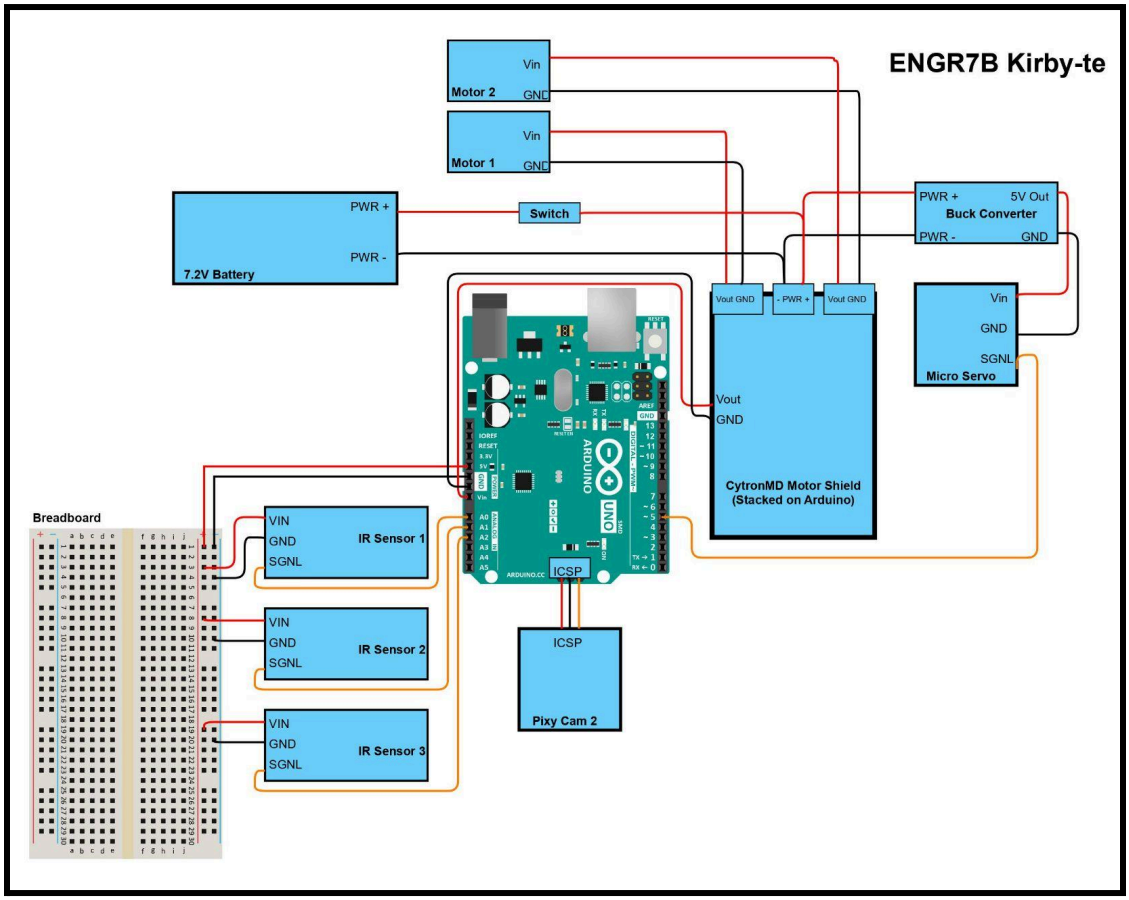
Isometric View: PixyCam Bracket



This bracket was designed to secure the PixyCam in its designated place in an upright orientation. In addition, it provided support to the PixyCam to keep its field of view steady through the rover's motion. The bracket has a height of 0.75 in, width of 1 in, and length of 1 in. The screw

holes are all M3, and are 0.61 inches apart to fit the dimensions of the PixyCam. It was secured onto two standoffs to attach it to the chassis.

Wiring Diagram



Algorithm Design

When considering the different possible turning angles, our team decided to position the three IR sensors in a triangular formation. Each IR sensor was attached to a horizontal slit to enable horizontal position adjustments to change the overall shape of the triangle if needed. With the IR sensors aligned as a triangle, our rover could track and follow all degrees of turns, both sharp and wide turns. With this design, our rover's center of rotation would be positioned at the wheel that is stationary when turning to keep the rover in line with the track afterwards, reducing zigzagging. By placing the base of the IR triangle collinear with the midpoint of the rover's length, the rover could be perfectly parallel with the track after most turns.

The algorithm was divided into three different modes, consisting of line following, can-grabbing, and braking. For line following, eight different IR sensor actuation scenarios were

considered. If only the middle sensor was “high,” the rover would go forward. If all three IR sensor values were “low,” the rover would slow down and detect the next turn. If only the right IR sensor was “high,” then the rover would turn right. If only the left IR sensor was “high,” then the rover would turn left. If both the middle and right IR sensors were “high,” the rover would do a sharp right turn. If both the middle and left IR sensors were” high,” the rover would do a sharp left turn. Lastly, if either all three IR sensors or just the left and right IR sensors detected “high,” the rover would brake and go into can-grabbing mode.

In can-grabbing mode, the rover was designed to initially rotate left in place until a can of the right color signature was detected. After the can was detected, the rover would adjust speeds until the can was in reach of the claws, when the micro servo would be activated to grab the can. Afterwards, the rover would go forward to move the can out of the designated box and enter the last mode, where the rover would brake and remain at rest until the Arduino was reset.

Pseudocode:

```
INCLUDE libraries necessary for the servo, motors, and Pixycam

SetUp:
    SET pins for the IR sensor, motors, and servo
    SET Pixycam parameters, Pixycam variables, IR sensor values, motor speeds, and enter
    mode 0
    ATTACH servo initialize Pixycam
Loop:

    Mode 0:
        READ all three IR Sensor values
            IF only the middle IR sensor detects high, THEN go forward
            ELSE IF all three IR sensors detect low, then go into detection mode

            turnDetect:
                IF only the left IR sensor detects high, THEN turn left
                IF only the right IR sensor detects high, THEN turn right

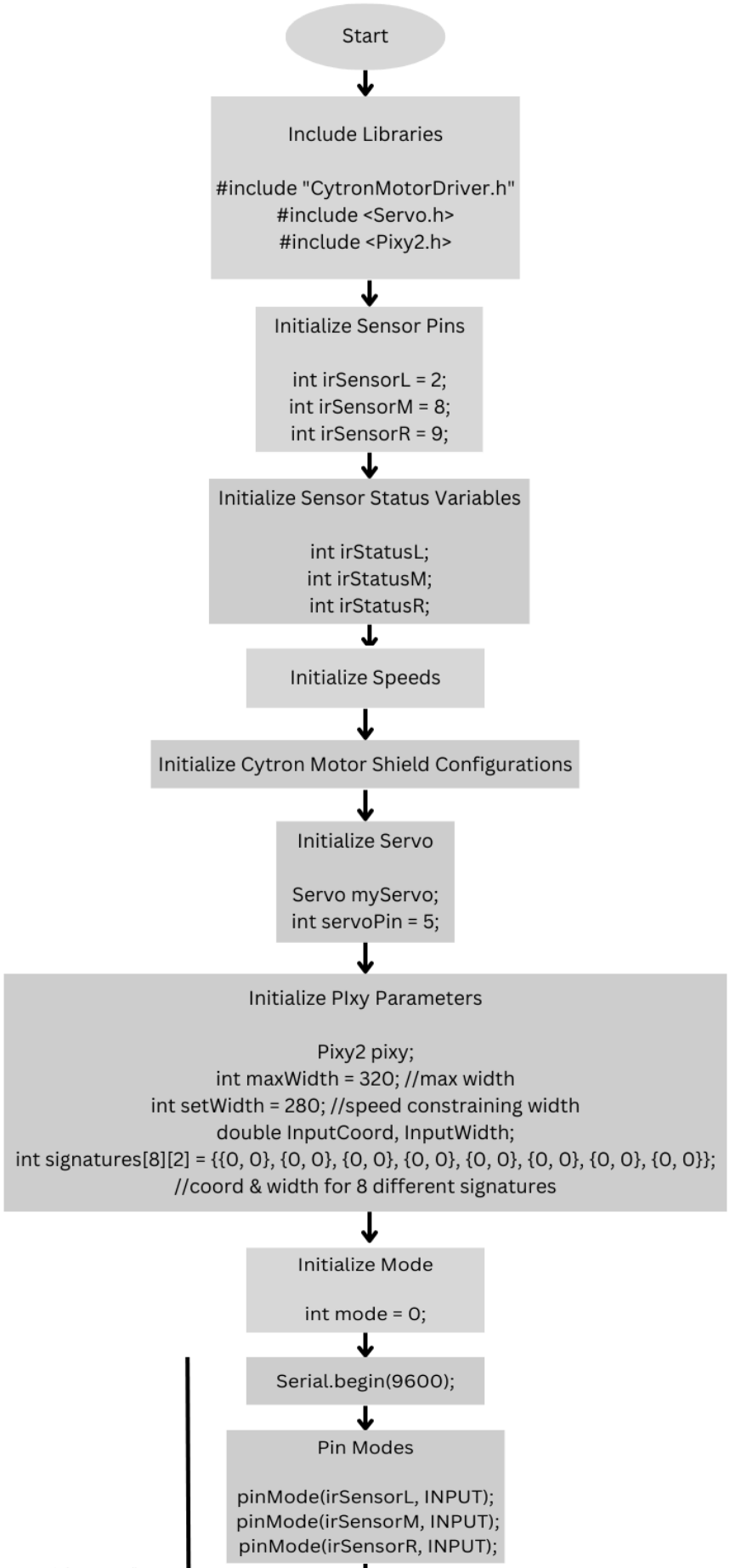
        ELSE IF all three IR sensors detect high OR only the left and right IR sensor detect
        high, THEN brake and go into mode 1

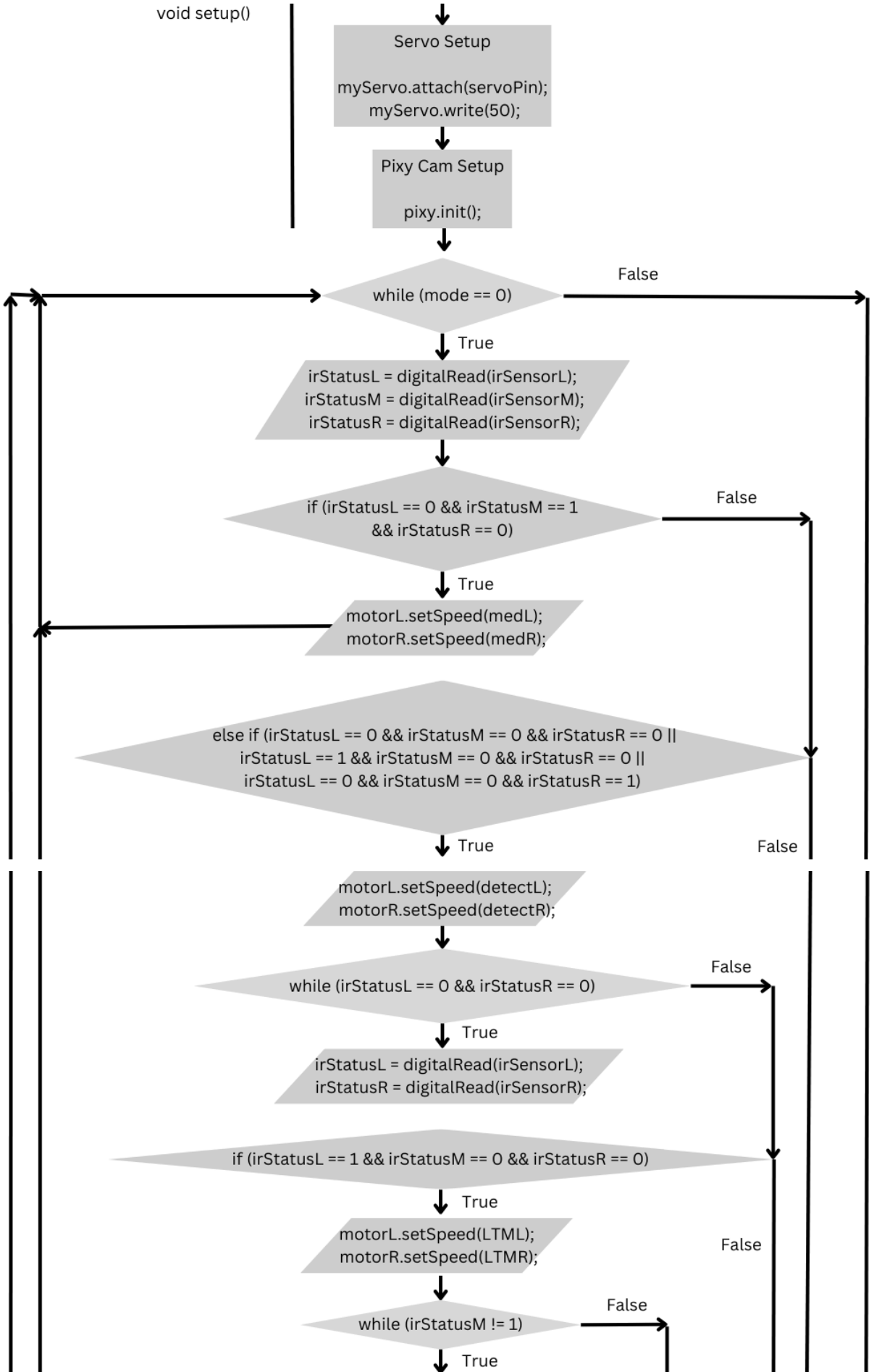
    Mode 1:
        GET blocks
        IF signatures are detected
            LOOP through each signature
```

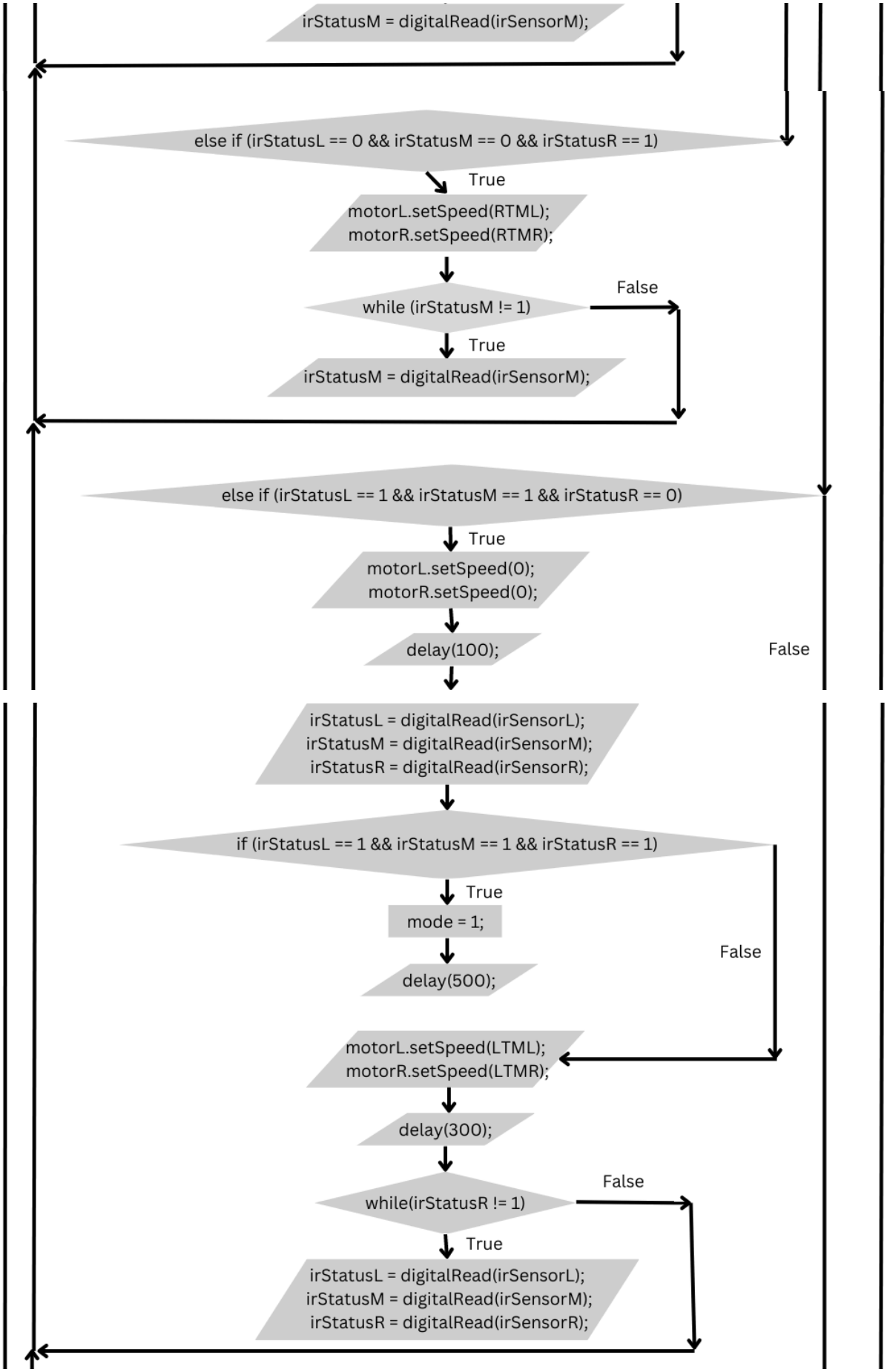
IF starting x coordinate and width values for each signature have not been initialized in an array
SET x coordinate and width values
READ current x coordinate and width values
IF current width is greater than 150
SET motor speeds to 0 and servo to 85
SET motor speeds to 255 for a bit
SET motor speeds to 0 and servo to 0
SET mode to 2
ELSE IF current x coordinate is greater than 140 and less than 150 and current width is not greater than 150
SET left motor speed by mapping current width to possible motor speeds (same)
SET right motor speed by mapping current width to possible motor speeds (same)
ELSE IF current x coordinate is greater than 150
SET left motor speed by mapping current width to possible motor speeds (less)
SET right motor speed by mapping current width to possible motor speeds (more)
ELSE IF current x coordinate is less than 140
SET left motor speed by mapping current width to possible motor speeds (more)
SET right motor speed by mapping current width to possible motor speeds (less)
ELSE WHILE blocks are not detected THEN turn rover clockwise

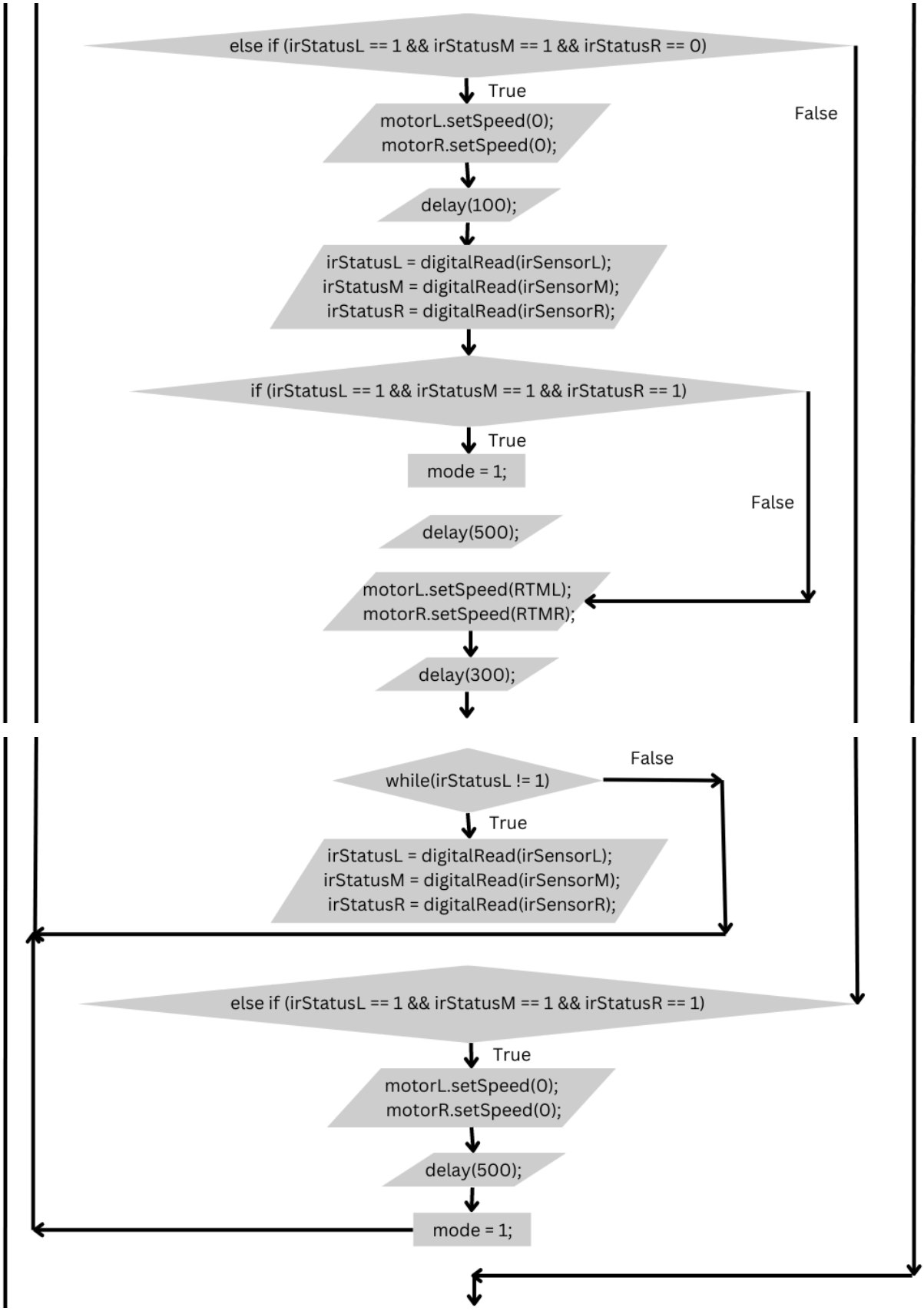
Mode 2:
SET motor speeds to 0

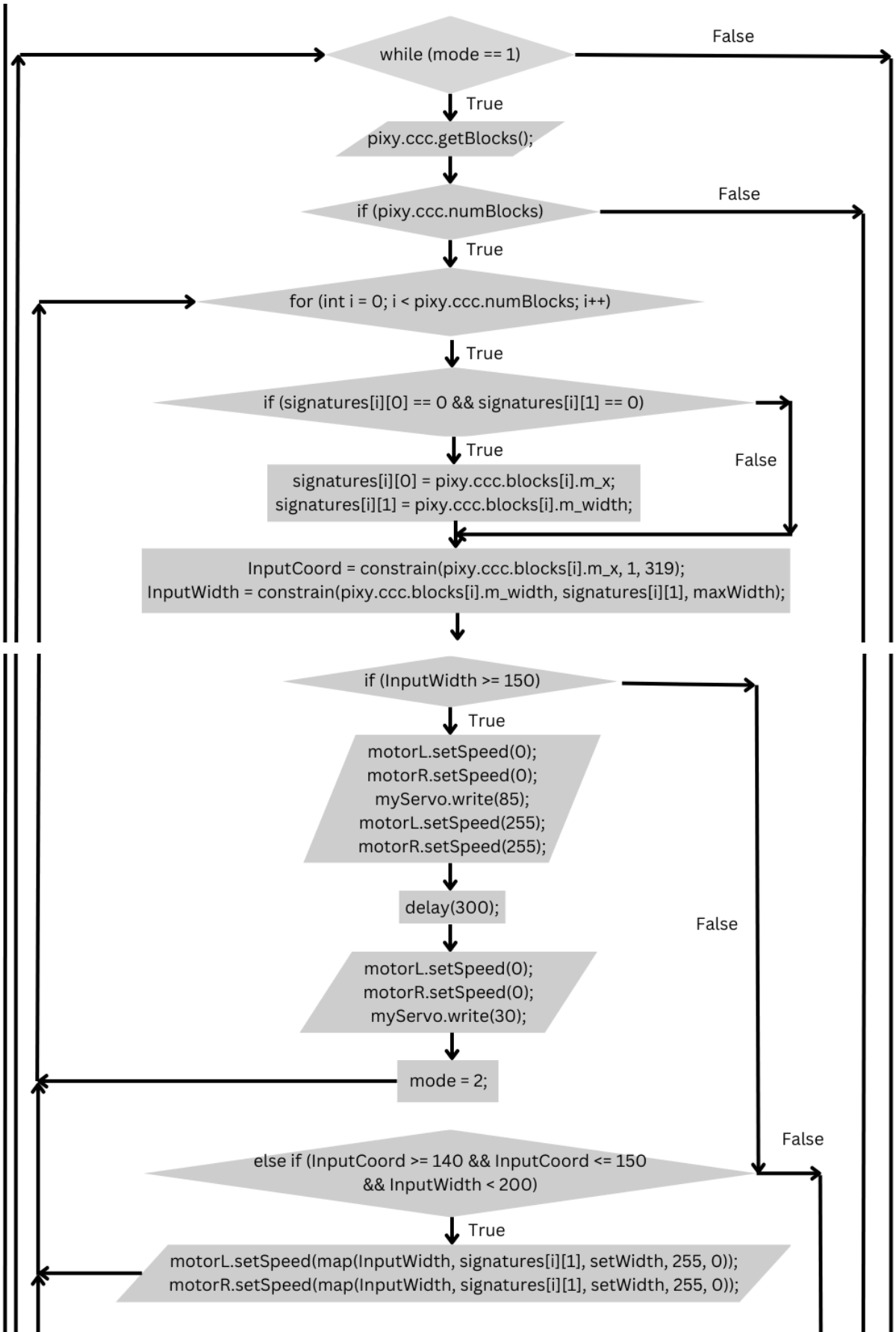
Flow Chart:

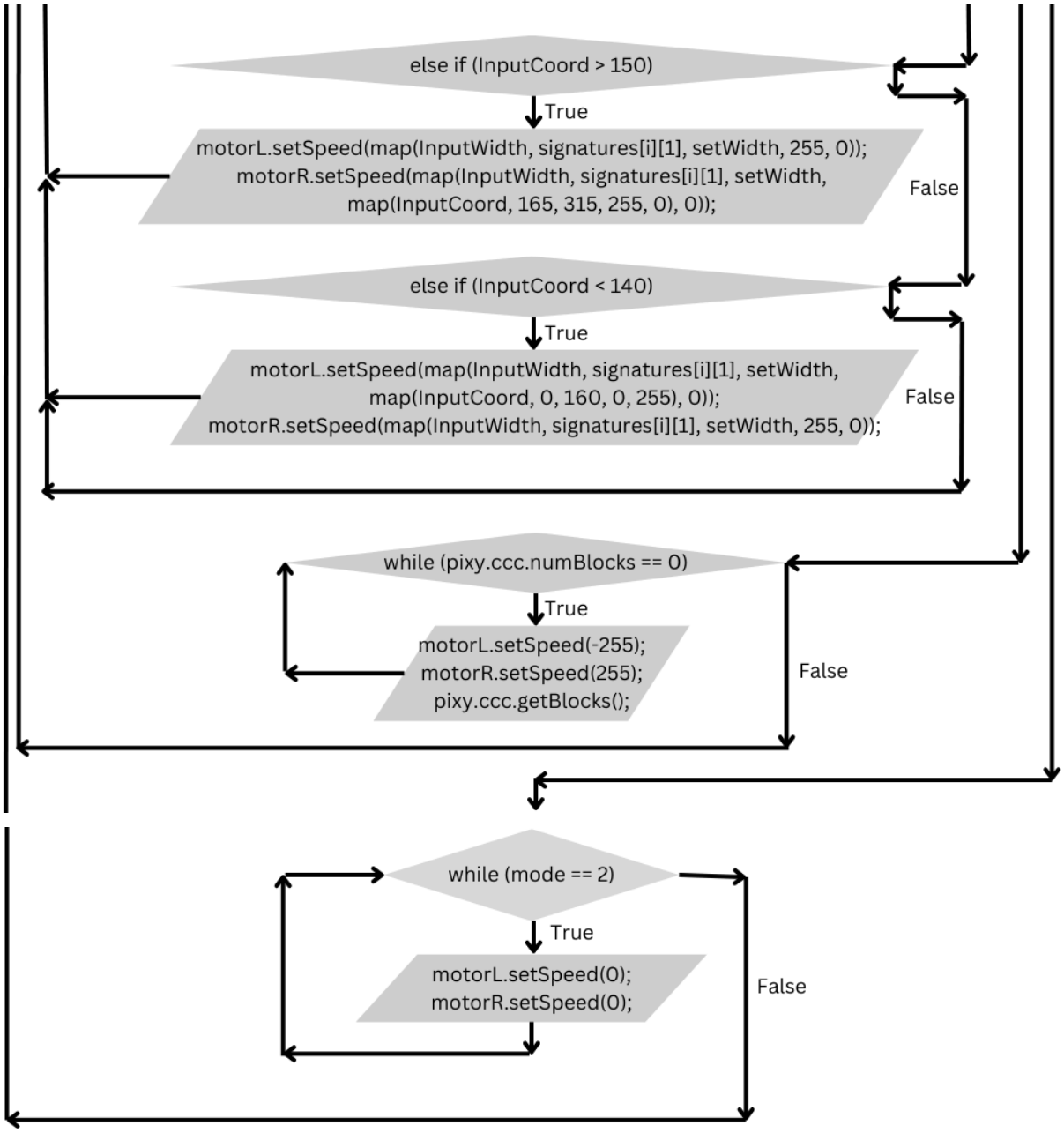

Start
Include Libraries
#include "CytronMotorDriver.h"
#include <Servo.h>
#include <Pixy2.h>
Initialize Sensor Pins
int irSensorL = 2;
int irSensorM = 8;
int irSensorR = 9;
Initialize Sensor Status Variables
int irStatusL;
int irStatusM;
int irStatusR;
Initialize Speeds
Initialize Cytron Motor Shield Configurations
Initialize Servo
Servo myServo;
int servoPin = 5;
Initialize PIxy Parameters
Pixy2 pixy;
int maxWidth = 320; //max width
int setWidth = 280; //speed constraining width
double InputCoord, InputWidth;
int signatures[8][2] = {{0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}};
//coord & width for 8 different signatures
Initialize Mode
int mode = 0;
Serial.begin(9600);
Pin Modes
pinMode(irSensorL, INPUT);
pinMode(irSensorM, INPUT);
pinMode(irSensorR, INPUT);


void setup()
Servo Setup
myServo.attach(servoPin);
myServo.write(50);
Pixy Cam Setup
pixy.init();
while (mode == 0)
False
True
irStatusL = digitalRead(irSensorL);
irStatusM = digitalRead(irSensorM);
irStatusR = digitalRead(irSensorR);
if (irStatusL == 0 && irStatusM == 1
&& irStatusR == 0)
False
True
motorL.setSpeed(medL);
motorR.setSpeed(medR);
else if (irStatusL == 0 && irStatusM == 0 && irStatusR == 0 ||
irStatusL == 1 && irStatusM == 0 && irStatusR == 0 ||
irStatusL == 0 && irStatusM == 0 && irStatusR == 1)
True
False
motorL.setSpeed(detectL);
motorR.setSpeed(detectR);
while (irStatusL == 0 && irStatusR == 0)
False
True
irStatusL = digitalRead(irSensorL);
irStatusR = digitalRead(irSensorR);
if (irStatusL == 1 && irStatusM == 0 && irStatusR == 0)
True
motorL.setSpeed(LTML);
motorR.setSpeed(LTMR);
False
while (irStatusM != 1)
False
True


irStatusM = digitalRead(irSensorM);
else if (irStatusL == 0 && irStatusM == 0 && irStatusR == 1)
True
motorL.setSpeed(RTML);
motorR.setSpeed(RTMR);
while (irStatusM != 1)
False
True
irStatusM = digitalRead(irSensorM);
else if (irStatusL == 1 && irStatusM == 1 && irStatusR == 0)
True
motorL.setSpeed(0);
motorR.setSpeed(0);
delay(100);
False
irStatusL = digitalRead(irSensorL);
irStatusM = digitalRead(irSensorM);
irStatusR = digitalRead(irSensorR);
if (irStatusL == 1 && irStatusM == 1 && irStatusR == 1)
True
mode = 1;
delay(500);
False
motorL.setSpeed(LTML);
motorR.setSpeed(LTMR);
delay(300);
while(irStatusR != 1)
False
True
irStatusL = digitalRead(irSensorL);
irStatusM = digitalRead(irSensorM);
irStatusR = digitalRead(irSensorR);


else if (irStatusL == 1 && irStatusM == 1 && irStatusR == 0)
True
False
motorL.setSpeed(0);
motorR.setSpeed(0);
delay(100);
irStatusL = digitalRead(irSensorL);
irStatusM = digitalRead(irSensorM);
irStatusR = digitalRead(irSensorR);
if (irStatusL == 1 && irStatusM == 1 && irStatusR == 1)
True
mode = 1;
False
delay(500);
motorL.setSpeed(RTML);
motorR.setSpeed(RTMR);
delay(300);
False
while(irStatusL != 1)
True
irStatusL = digitalRead(irSensorL);
irStatusM = digitalRead(irSensorM);
irStatusR = digitalRead(irSensorR);
else if (irStatusL == 1 && irStatusM == 1 && irStatusR == 1)
True
motorL.setSpeed(0);
motorR.setSpeed(0);
delay(500);
mode = 1;


while (mode == 1)
False
True
pixy.ccc.getBlocks();
if (pixy.ccc.numBlocks)
False
True
for (int i = 0; i < pixy.ccc.numBlocks; i++)
True
if (signatures[i][0] == 0 && signatures[i][1] == 0)
True
False
signatures[i][0] = pixy.ccc.blocks[i].m_x;
signatures[i][1] = pixy.ccc.blocks[i].m_width;
InputCoord = constrain(pixy.ccc.blocks[i].m_x, 1, 319);
InputWidth = constrain(pixy.ccc.blocks[i].m_width, signatures[i][1], maxWidth);
if (InputWidth >= 150)
True
motorL.setSpeed(0);
motorR.setSpeed(0);
myServo.write(85);
motorL.setSpeed(255);
motorR.setSpeed(255);
delay(300);
False
motorL.setSpeed(0);
motorR.setSpeed(0);
myServo.write(30);
mode = 2;
False
else if (InputCoord >= 140 && InputCoord <= 150
&& InputWidth < 200)
True
motorL.setSpeed(map(InputWidth, signatures[i][1], setWidth, 255, 0));
motorR.setSpeed(map(InputWidth, signatures[i][1], setWidth, 255, 0));


else if (InputCoord > 150)
True
motorL.setSpeed(map(InputWidth, signatures[i][1], setWidth, 255, 0));
motorR.setSpeed(map(InputWidth, signatures[i][1], setWidth,
map(InputCoord, 165, 315, 255, 0), 0));
False
else if (InputCoord < 140)
True
motorL.setSpeed(map(InputWidth, signatures[i][1], setWidth,
map(InputCoord, 0, 160, 0, 255), 0));
motorR.setSpeed(map(InputWidth, signatures[i][1], setWidth, 255, 0));
False
while (pixy.ccc.numBlocks == 0)
True
motorL.setSpeed(-255);
motorR.setSpeed(255);
pixy.ccc.getBlocks();
False
while (mode == 2)
True
motorL.setSpeed(0);
motorR.setSpeed(0);
False

Action Item Report

Task Assignment

Considering the complexity of the task at hand, it was necessary to delegate tasks between team members and have each team member specialize in different aspects of the rover. The initial task assignment process was simple, and involved each team member selecting what positions they felt most comfortable with and had the most experience in. Furthermore, given that certain tasks occur at different times in the project, like the workload for CAD being more towards the beginning of the project, positions were selected to ensure that everyone contributed equally throughout the project. The assignments we settled on were Eric and Gavin on algorithm design and coding, Victor and Kristen on fabrication and electronics, and Angel and Caden on CAD design. Weekly tasks were fluid, and team members helped with pressing tasks even if it wasn't within their initial job description.

Gantt Chart

Kirby-Te					Planned		Actual		Due Date		Overlap			
Tuesday, 2PM	Planned		Actual		Week 1		Week 2		Week 3		Week 4		Week 5	
Activity	Start	End	Start	End										
Team Formation	1	1	1	1										
Team Name & Captain Chosen	1	1	1	1										
Rover Design	2	4	2	5										
Chassis Ideation/CAD	2	3	2	4										
Material Selection (Wheel & Base)	2	4	2	3										
Claw Design	2	4	2	4										
SolidWorks Drawings	2	4	3	4										
Purchase Order Form	3	4	3	4										
Algorithm Design & Coding	2	6	3	9										
Preliminary Algorithm Design	2	6	3	4										
Line Following Code	2	6	3	9										
PixyCam Code	4	6	4	7										
Fabrication/Electronics	6	8	5	7										
Wiring Diagram	4	5	4	5										
Laser Cutting & 3D Printing	6	7	5	7										
Full Structural Assembly	7	7	6	7										
Soldering & Electronics Integration	6	7	5	7										
Testing	7	10	7	10										
Test Electronics	7	7	7	7										
Tune Arduino Line Following	8	10	7	10										
Test Claw Mechanism	8	8	7	7										
Final Test and Evaluation	9	10	9	10										
FINAL COMPETITION	10	10	10	10										
Action Item Reports	3	10	3	10										
Preliminary Presentation	6	6	6	6										
Final Presentation	Finals	Finals	Finals	Finals										



Evaluation

Calculations

Vehicle Parameters:

- Total mass (with battery): 1026 grams
- Predicted Drive Time: 20 minutes (minimum)
- Mechanical Advantage: 34:1
- Stall Torque: 6.8 $kg \bullet cm$ at 6V
- Stall Weight: 1.7 kg at 6V

Elaboration:

To obtain the predicted drive time, sum the total current draw of the electronics on the rover and use the equation, $I = \frac{\Delta q}{\Delta t}$ to solve for Δt, the total run time of the rover. In this case, the current draw from the components will be 140 mA from the PixyCam, a maximum of 5A per motor, 220 mA from the micro servo, and a maximum current draw of 40 mA per port on the Arduino, with 5 connections total. Summing these currents results in a worst-case current draw of 10,560 mA.

Substituting this value into the equation for I, and substituting the charge of the battery, 3600 mAh, for Δq yields a minimum operating time of 0.34 hours, or approximately 20 minutes.

To determine the mechanical advantage of the motor, simply take the output torque and divide it by the input torque. Given that $\tau_{output} = \frac{N_{output}}{N_{input}} \tau_{input}$, rearranging the equation yields $\frac{\tau_{output}}{\tau_{input}} = \frac{N_{output}}{N_{input}}$. Therefore, the mechanical advantage can be taken from the manufacturer as 34 to 1 based on the gear ratio.

The stall torque is given by the manufacturer for the 34:1 gearmotor as 11 $kg \bullet cm$ for operation at 12V. Given that our motors will operate at 7.2V, based on the battery, it will have a stall torque that is lower than this. At 6V, the stall torque is listed at 6.8 $kg \bullet cm$. Therefore, the actual stall torque of the 34:1 gearmotor will be somewhere in between these values. We will take the lower stall torque value for our calculations. By comparison, the stall torque for the 47:1 gearmotor will be 15 $kg \bullet cm$ at 12V, and 9.1 $kg \bullet cm$ at 6V. Despite its lower stall torque, the 34:1 gearmotor was chosen because it will rotate with a higher speed than the 47:1 gearmotor. It was decided that since the motor would not need a large amount of torque to maneuver the rover, speed was more important than torque for the purposes of the competition. For example, at 6V, the 34:1 gearmotor has a no-load RPM of 290, while the 47:1 gearmotor has only 210 RPM.

To calculate the stall weight of the motor, you find the maximum weight that the motor can lift with a pulley with the same radius as the selected wheel, 40mm. Since $\tau = Fr$ when the force is applied perpendicular to the radius, the torque of the motor will be the stall torque multiplied by the radius of the wheel. However, the units that the stall torque is given in by the manufacturer, $kg \bullet cm$, is not conventional. A stall torque of 6.8 $kg \bullet cm$ is the same as the torque produced by a 6.8 kg weight attached 1 cm away from the axis of rotation. Therefore, taking the worst-case torque value at 6V, 6.8 $kg \bullet cm$, and dividing it by the radius of 4cm yields a stall weight of 1.7 kg for one motor. The mass of our rover is 1.026 kg, so the stall weight of one motor is greater than the total mass, meaning that there will be plenty of torque to move the rover through the competition track.

Test Plan

Rover Parts:

- Motor Direction:
    1. Connect mounted motors to proper ports on the motor shield with soldered wiring.
    2. Ensure H Bridges on Cytron Motor Shield are placed in the correct positions based on the code written.
    3. Using manual buttons on the motor shield, test DIR1 and DIR2 for each motor, ensuring they are turning as intended. If not, switch motor wiring.
    4. Test motors with code to ensure Arduino connectivity.

- Buck Converter Calibration:
    1. Solder wires to each end of the buck converter, connect Voltage in and Ground in wire to the motor shield.
    2. Using a multimeter, read voltage going through the buck converter when connected to a 7.2V Battery, calibrate potentiometer using flathead screwdriver until multimeter reads 5V out.

- Servo Claw Functionality:
    1. Connect servo power to the buck converter, ensuring the buck converter is outputting 5V.
    2. Connect servo signal pin to pin 5 on Arduino, as specified in code.
    3. Mount 3D printed claw piece on claw using provided servo horn.
    4. Test position 0 for servo, then using Arduino set desired position for resting point on claw.
    5. Input multiple servo positions, observe claw functionality.
    6. Connect the second claw arm, ensuring spur gear teeth are meshing properly.
    7. Test servo inputs, actuating claw, and set proper servo values for desired claw position.

- Wheel Alignment:
    1. Attach wheels to motors once motors are mounted on the chassis.
    2. Actuate motors, observing the wheel parallel to the plane, and see whether they move completely straight or oscillate.
    3. If oscillating, adjust wheels to be completely parallel to the motor.
    4. Repeat until both wheels are aligned.

- IR Sensor Power:
    1. To ensure all IR sensors receive power, test the soldered bus using a multimeter.
    2. Test from all portions of the bus, including opposite ends of soldered wire, to ensure current passes through the bus.
    3. Insulate with electrical tape.

Arduino Parts:

- IR Sensors Sensitivity:
    1. Gently bend the sensor pins perpendicular to the IR Sensor and facing towards the ground.
    2. Make sure IR sensors are in the right pins and receive power, indicated by one red LED light on.
    3. Connect the USB cord to a laptop and the Arduino, then compile and upload the file named “ir sensor”, which prints the signals of the three IR sensors every second.
    4. Using a mini screwdriver, turn the potentiometers, clockwise making the sensors more sensitive and counterclockwise making the sensors less sensitive.

5. Position the rover on the preliminary course so that one IR sensor is on the black tape at a time. Then, open the serial monitor on the Arduino IDE, and check to see values of each IR sensor. Ensure that the sensor reads “1” when on top of the black tape and “0” on a white surface. Repeat steps 4-5 for all IR sensors until functioning as specified.

- Line Following:
    - Wide turns:
        1. Check to see that all IR sensors are working as intended. See IR Sensors Sensitivity if IR sensors need to be recalibrated.
        2. Connect the USB cord to a laptop and the arduino, then compile and upload the file named “Kirbyte_Final”
        3. Place the rover on the track, aligning the middle sensor on top and parallel to the black line.
        4. Check to see if turns work as intended. If not, below is a list to troubleshoot:
        a. If the rover is missing the turn, then turn the forward speed and/or delay of detect() down.
        b. If the rover is overturning, then turn the turn speed down.
        c. If the rover is zigzagging too much, then turn forward and turn speed down.
        d. If the rover turns the opposite way as designed, then turn the forward and turn speed down to reduce zigzagging.
        e. If the rover does not detect the brake, then turn the forward speed and/or delay of detect() down.

    - Sharp turns:
        1. Check to see that all IR sensors are working as intended. See IR Sensors Sensitivity if IR sensors need to be recalibrated.
        2. Connect the USB cord to a laptop and the arduino, then compile and upload the file named “Kirbyte_Final_Code_Sharp_Turns”.
        3. Place the rover on the track, aligning the middle sensor on top and parallel to the black line.
        4. Check to see if turns work as intended. If not, below is a list to troubleshoot:
        a. If the rover is missing the turn, then turn the forward speed and/or delay of detect() down. Also, consider increasing the delay of brake() in the sharpleft and sharpright functions and decreasing the detect speed.
        b. If the rover is overturning, then turn the turn speed down.
        c. If the rover is zigzagging too much, then turn forward and turn speed down.
        d. If the rover turns the opposite way as designed, then turn the forward and turn speed down to reduce zigzagging. Also, make sure that the rover is aligned properly in the beginning.
        e. If the rover does not detect the brake, then turn the forward speed down, increase the delay of the brake(), and decrease the delay of detect().

- Object Detection:
    1. Elevate the rover by placing a box underneath.
    2. Hold an empty soda can wrapped in pink construction paper such that the PixyCam is able to detect it.
    3. Move the can side to side to test motor speeds.
        a. When facing the PixyCam camera, holding the can to its left should cause the left motor to have a low (or 0) rpm and the right motor to have a high rpm
        b. Vice versa, holding the can to the right of the camera should cause the right motor to have low (or 0) rpm and the left motor to have a high rpm
        c. The differences in rpm should be greater the further left/right the can is held.
    4. If the motor speeds are not proportional to its x-coordinate in the camera's field of view, edit the motor mapping code to reflect the desired rpm's.
    5. Move the can forward and back to test motor speeds.
        a. When the can is close to the camera, both motors should have low rpm
        b. When the distance between the camera and the can is increased, the motor rpm's should increase
    6. If the motor speeds are not proportional to the width in the camera's field of view, edit the motor mapping code to reflect the desired rpm's.
    7. Place the can to the grabbing distance.
        a. If the servo closes the claw when the can is too far from the camera or too close to the camera, adjust the constraining width values.
    8. Once satisfied, initiate object detecting mode and place the rover on the ground far away from the can to observe its dynamics. Adjust code accordingly.

- Combined:
    1. Place the rover on a track made from black tape on white poster board with a pink soda can a few feet in front of the end.
    2. Turn the rover on and observe its turning, speeds, and overall performance.
    3. Adjust turning speeds and straight line speeds to balance line following turning precision and speed to minimize the time spent traversing the course.
    4. Adjust motor mapping to minimize time spent traveling to the soda can, grabbing the can, and pushing the can out of the box.
    5. Repeat steps 1-4 to get the best time for completing the objective.

Results & Discussion

Following the final competition, our team received 2nd place overall, out of 40 teams. Our rover was able to complete the necessary objectives of line following and claw actuation, all in a time of 20.5 seconds.

The rover structure was made of poplar wood, laser cut to meet design requirements, staying within the 12x16" dimensions specified. The structure of the rover proved to be durable,

even after crashing into the wall many times, and handled the load of the electronics it was meant to carry. The rigidity of the structure was sufficient, with all connections done via steel screws and standoffs. The rover was stable during operation, with properly aligned wheels which allowed line following to be performed effectively.

The weight of the rover was minimized, as there were no unnecessary portions which increased the weight. Additionally, motors with lower stall torque were used, as it allowed for a higher RPM, increasing the maximum speed of the rover. This was chosen because the rover stayed on a flat surface the entire time, therefore the torque necessary to propel it was not too great.

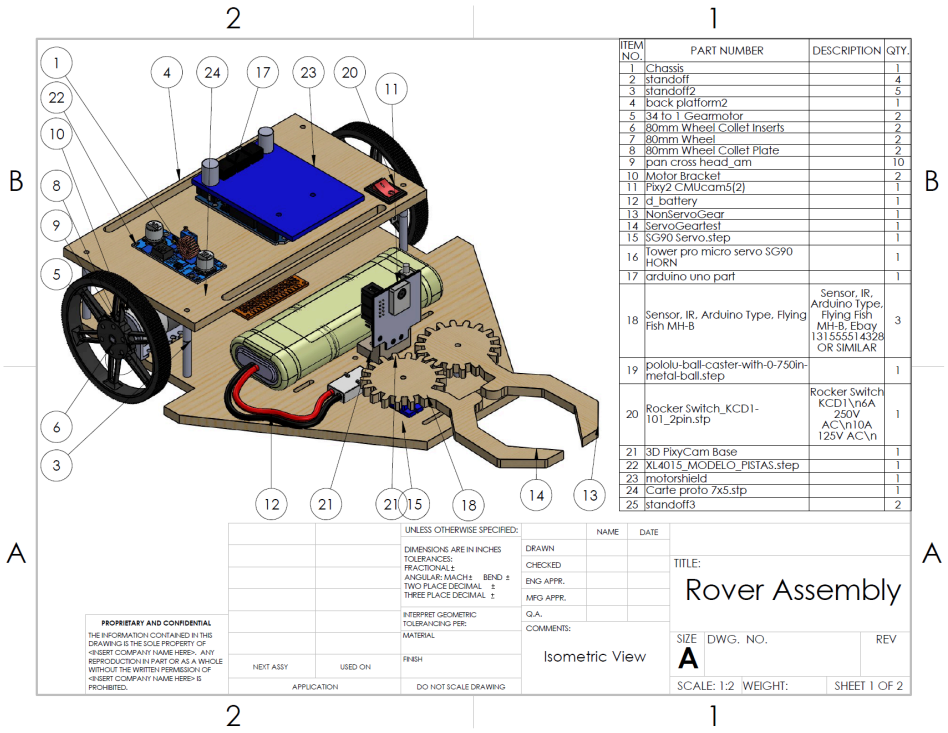
The agility of the rover was sufficient, able to perform stationary turns, and the IR sensor positioning allowed for incredibly sharp turns to be possible.

Throughout the quarter, the team learned that communication is incredibly important, both in the engineering design process as well as for all other events, as it leads to an environment which is much more conducive to faster iteration and development of a design. Additionally, willingness to ask questions was greatly helpful, as design validation and new ideas concerning code were advised for or against, while also learning from the experience of those who have taken the course in the past. Another possible improvement which would be incorporated given an opportunity to work on the project again is to be willing to try unconventional ideas, as well as clarifying the design requirements, as it could lead to possible changes which would be beneficial, such as a PixyCam which could help in line following with PD control.

Overall, the ENGR 7B autonomous rover project was a great learning experience and opportunity to develop several skills as an engineer, and the success which came out of it was a result of a strong team which came together, shared ideas collaboratively, and stayed on task throughout the quarter to achieve a common goal, with a desire for strong performance.

Appendix A: SOLIDWORKS Drawings

Drawing: Isometric View KIRBY-TE Rover Assembly CAD



ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	Chassis		1
2	standoff		4
3	standoff2		5
4	back platform2		1
5	34 to 1 Gearmotor		2
6	80mm Wheel Collet Inserts		2
7	80mm Wheel		2
8	80mm Wheel Collet Plate		2
9	pan cross head_am		10
10	Motor Bracket		2
11	Pixy2 CMUcam5(2)		1
12	d_battery		1
13	NonServoGear		1
14	ServoGeartest		1
15	SG90 Servo.step		1
16	Tower pro micro servo SG90 HORN		1
17	arduino uno part		1
18	Sensor, IR, Arduino Type, Flying Fish MH-B	Sensor, IR, Arduino Type, Flying Fish MH-B, Ebay 131555514328 OR SIMILAR	3
19	pololu-ball-caster-with-0-750in-metal-ball.step		1
20	Rocker Switch_KCD1-101_2pin.stp	Rocker Switch KCD1\n6A 250V AC\n10A 125V AC\n	1
21	3D PixyCam Base		1
22	XL4015_MODELO_PISTAS.step		1
23	motorshield		1
24	Carte proto 7x5.stp		1
25	standoff3		2

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
NEXT ASSY | USED ON
APPLICATION | DO NOT SCALE DRAWING

	NAME	DATE
DRAWN		
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		
COMMENTS:		

Isometric View

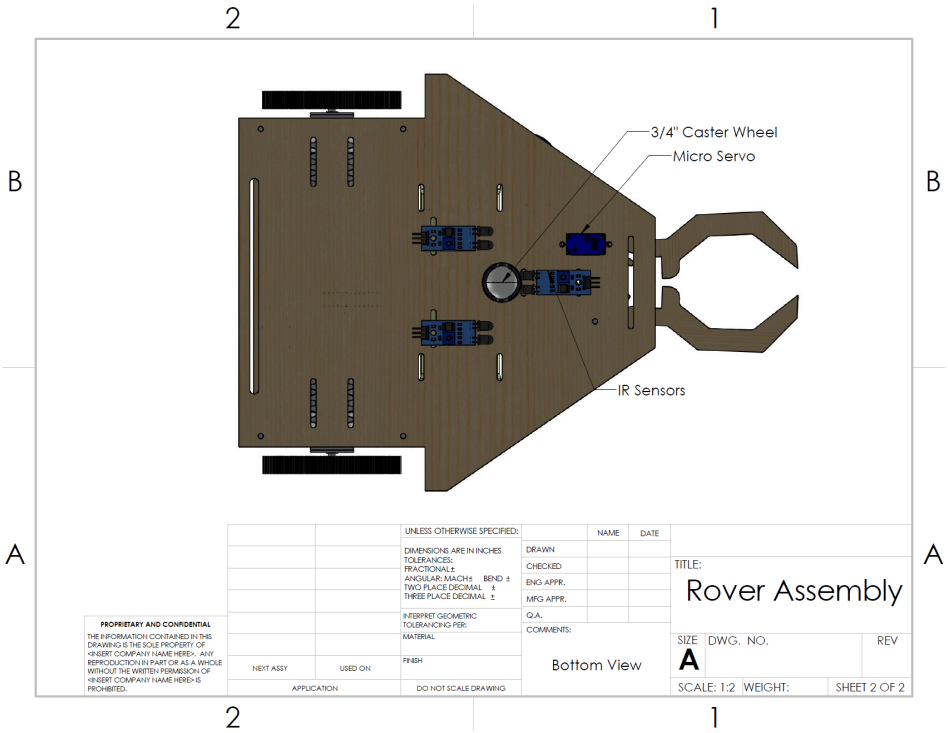
TITLE: Rover Assembly

SIZE A | DWG. NO. | REV

SCALE: 1:2 | WEIGHT: | SHEET 1 OF 2

PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.

Drawing: Bottom View KIRBY-TE Rover Assembly



UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
NEXT ASSY | USED ON
APPLICATION | DO NOT SCALE DRAWING

	NAME	DATE
DRAWN		
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		
COMMENTS:		

Bottom View

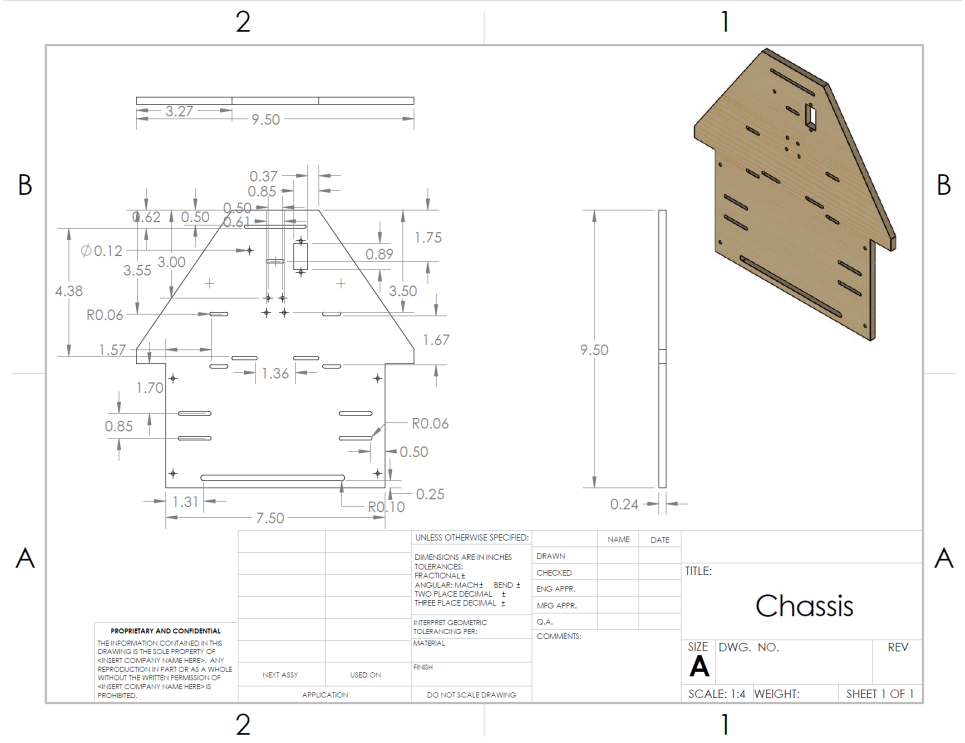
TITLE: Rover Assembly

SIZE A | DWG. NO. | REV

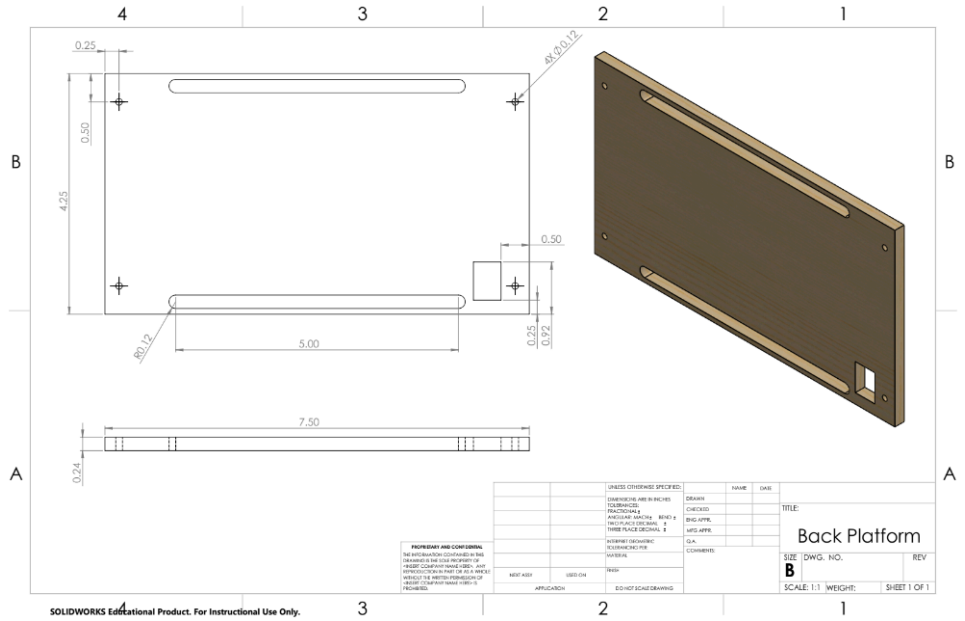
SCALE: 1:2 | WEIGHT: | SHEET 2 OF 2

PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.

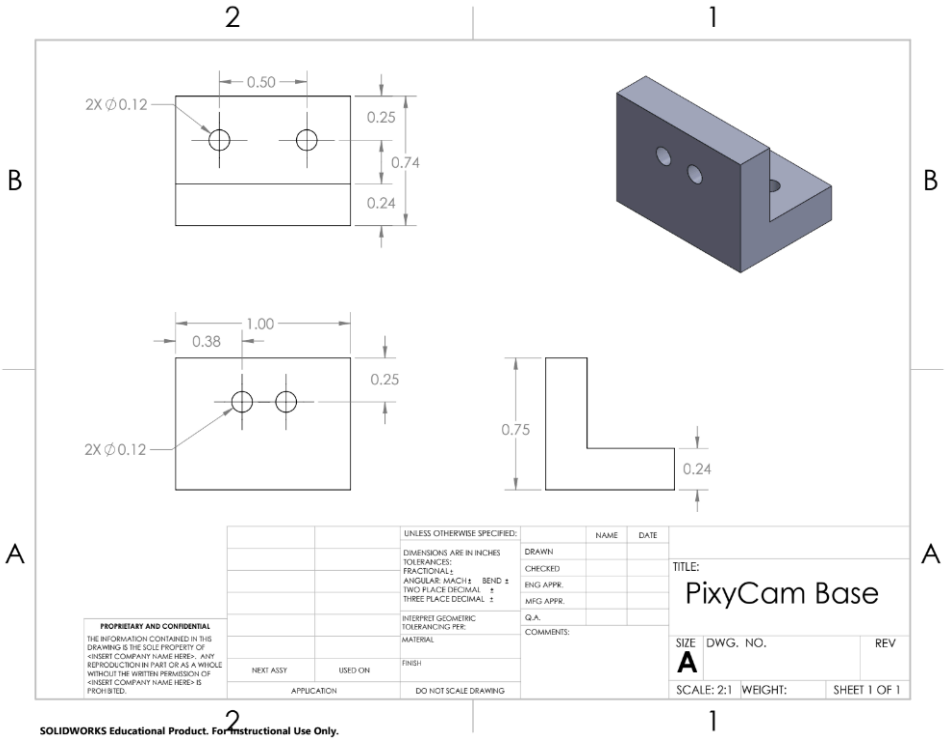
Drawing: Chassis CAD



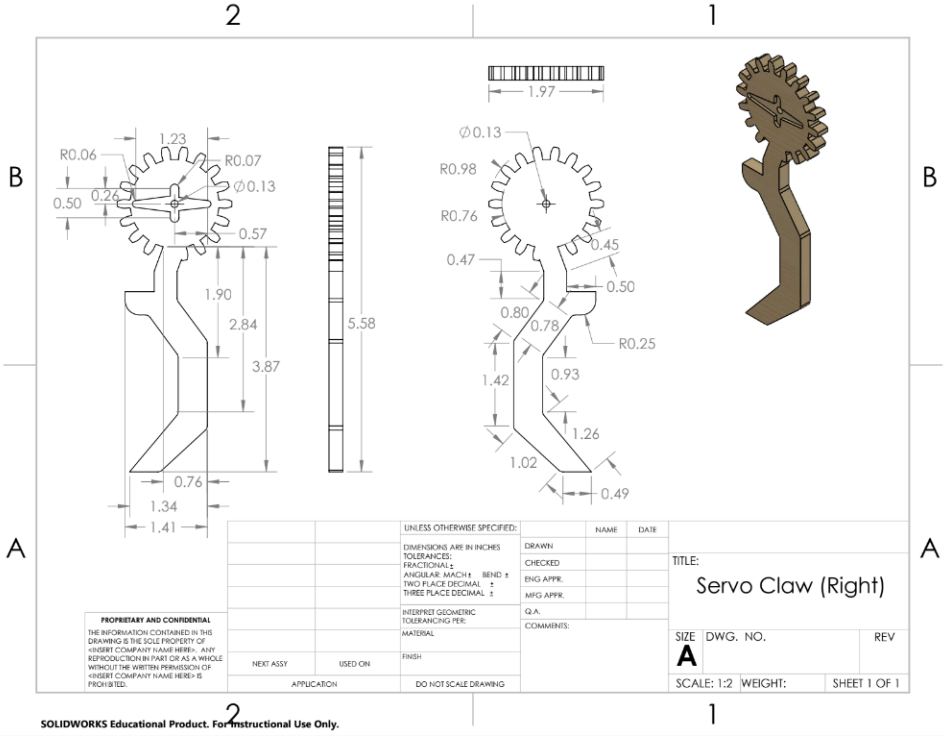
Drawing: Back Platform CAD



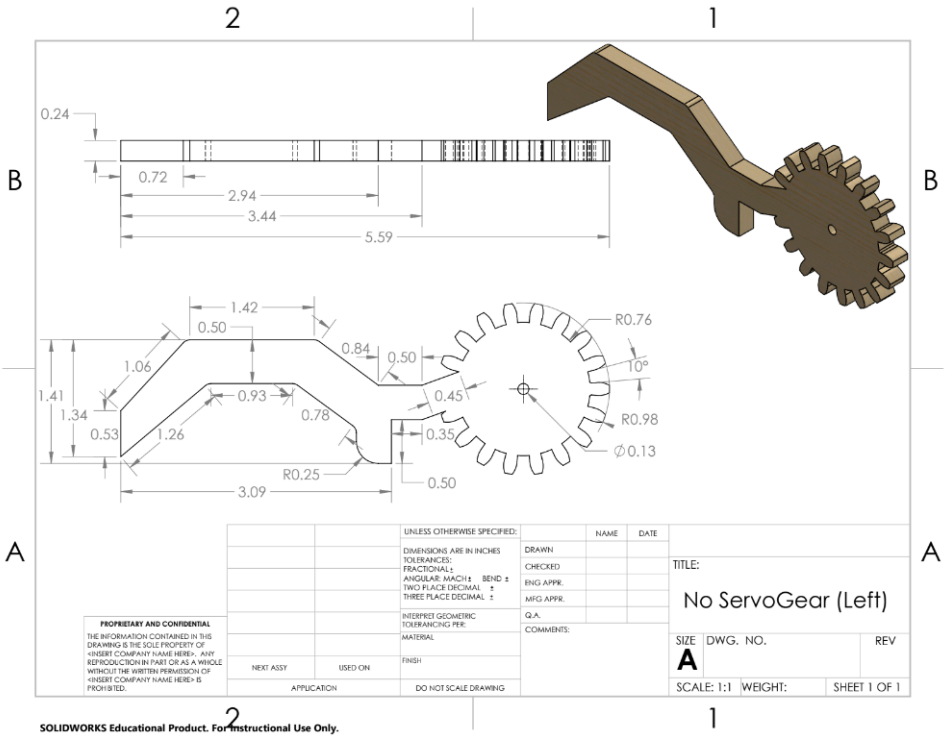
Drawing: PixyCam Base CAD



Drawing: Right Gear Claw CAD



Drawing: Left Gear Claw CAD



Appendix B: Bill of Materials

Quantity*	# of Units in package*	Company*	Item Description*	Catalog #*	Price*	Estimated Extended Price*
1	1	Arduino	Arduino Uno	A000066	$27.60	$27.60
1	1	Cytron	Motor Driver Shield	SHIELD-MDD10	$22.55	$22.55
1	4	Amazon	Buck Conveter	B079N9BFZC	$14.99	$3.75
1	3	Amazon	Male Tamiya Connector	B07VL2B5C8	$8.69	$2.90
1	10	Amazon	Power Switch	B071Y7SMVQ	$7.99	$0.80
1	2	Amazon	Battery	B07VLKP6RJ	$34.99	$17.50
1	1	Sparkfun/adafruit/robotshop	PixyCam 2	14678	$69.95	$69.95
3	20	Amazon	IR Sensors	B08DR1W3BK	$10.99	$1.65
1	12	Amazon	SG90 Microservo	B08FJ27Q1H	$20.99	$1.75
1	1	Amazon	Wire	B07SJ44SN1	$16.99	$16.99
2	1	Pololu	34:1 Gearmotor	3204	$28.95	$57.90
2	2	Pololu	80mm x 10mm Wheels	3690	$9.95	$9.95
1	1	Pololu	3/4" Caster Wheel	995	$3.95	$3.95
2	2	Pololu	Motor Bracket	2676	$7.95	$7.95
1	1	N/A	Italian Poplar 16"x16" (6mm thick)	N/A	$2.78	$2.78
N/A	N/A	N/A	3D Printing	N/A	$10.00	$10.00
N/A	N/A	N/A	Laser Cutting	N/A	$4.00	$4.00
					Subtotal	**$261.96**
				*Tax rate:	7.75%	$20.30
					TOTAL ORDER PRICE	**$282.26**

Appendix C: Arduino Code

```
/*****
Team Name: KIRBY-TE
Team Members: Caden Michael McCarthy, Gavin Nguyen, Eric Cheng, Victor Plesco, Angel Israel Derouin, Kristen Chung
Last Edit: 3/21/2023 11:02 AM
Code Purpose: This code is designed to enable a rover with three IR Sensors placed in a triangle to follow a black line, and if it detects black on all three sensors, then the rover will stop and
activate the cup grabbing code, which will find a cup, drive to it, and when in the range of the claw, will grab the can and go forward.
                *****/

//libraries
#include "CytronMotorDriver.h"
#include <Servo.h>
#include <Pixy2.h>

//sensor pins
int irSensorL = 2;
int irSensorM = 8;
int irSensorR = 9;

//sensor statuses
int irStatusL;
int irStatusM;
int irStatusR;

//line following speeds
int medL = 240;
int medR = 240;

//detect speeds
int detectL = 210;
int detectR = 210;

//left turn speeds
int LTML = 0;
int LTMR = 200;

//right turn speeds
```

```
int RTML = 200;
int RTMR = 0;

//cytron motor shield
CytronMD motorL(PWM_DIR, 3, 4); //motorL
CytronMD motorR(PWM_DIR, 6, 7); //motorR

//servo initiation & pins
Servo myServo;
int servoPin = 5;

//pixycam
Pixy2 pixy;
int maxWidth = 320;   //max width
int setWidth = 320;   //speed constraining width
double InputCoord, InputWidth;
int signatures[8][2] = {{0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}, {0, 0}};  //coord & width for 8 different signatures

//line following if 0, object detection if 1, stop if 2
int mode = 0;

void setup() {
 //serial monitor
 Serial.begin(9600);

 //pin setup for IR Sensors
 pinMode(irSensorL, INPUT);
 pinMode(irSensorM, INPUT);
 pinMode(irSensorR, INPUT);

 //servo setup
 myServo.attach(servoPin);
 myServo.write(0);   //initial position

 //pixy cam setup
 pixy.init();

}

void loop() {
```

```
 //line detection mode
 while (mode == 0) {

  //read sensor values
  detect();

  //forward
  if (irStatusL == 0 && irStatusM == 1 && irStatusR == 0) {
   forward();
  }

  //normal left and right turns - continue detecting
  else if(irStatusL == 0 && irStatusM == 0 && irStatusR == 0 || irStatusL == 1 && irStatusM == 0 &&
irStatusR == 0 || irStatusL == 0 && irStatusM == 0 && irStatusR == 1) {
    turnDetect();

    if (irStatusL == 1 && irStatusR == 0) {
     leftTurn();
    }

    else if (irStatusL == 0 && irStatusR == 1) {
     rightTurn();
    }
  }

  //brake and change to object detecting mode
  else if (irStatusL == 1 && irStatusM == 1 && irStatusR == 1|| irStatusL == 1 && irStatusM == 0 &&
irStatusR == 1) {
    brake();
    delay(100);
    myServo.write(50);
    mode = 1;
  }
 }

 //object detecting mode
 while (mode == 1) {

  //start pixy cam
  pixy.ccc.getBlocks();
```

```
  //if there are recognized colors
  if (pixy.ccc.numBlocks) {

    //iterate through each signature
    for (int i = 0; i < pixy.ccc.numBlocks; i++) {

      //store initial x coordinate and width
      if (signatures[i][0] == 0 && signatures[i][1] == 0) {
        signatures[i][0] = pixy.ccc.blocks[i].m_x;
        signatures[i][1] = pixy.ccc.blocks[i].m_width;
      }

      //most recent x coordinate and width
      InputCoord = constrain(pixy.ccc.blocks[i].m_x, 1, 319);
      InputWidth = constrain(pixy.ccc.blocks[i].m_width, signatures[i][1], maxWidth); //maxWidth =
340, setWidth can now be adjusted

      //if at grab width (150), brake and grab the can/cup
      if (InputWidth >= 150) {
        motorL.setSpeed(0);
        motorR.setSpeed(0);
        grab(); //initiates claw
        afterGrab();
        mode = 2; //stop everything and brake
      }

      //if can/cup is almost mid and width is too small - go zoom if still far away, go slow when almost
grab width
      else if (InputCoord >= 140 && InputCoord <= 150 && InputWidth < 200) {  //width changed from
200 to 150
        motorL.setSpeed(map(InputWidth, signatures[i][1], setWidth, 255, 0));
        motorR.setSpeed(map(InputWidth, signatures[i][1], setWidth, 255, 0));
      }

      //if can/cup is to the right - L motor based solely off width, R motor based off width with max
speed affected by mid/maxR x coord
      else if (InputCoord > 150) {
        motorL.setSpeed(map(InputWidth, signatures[i][1], setWidth, 255, 0));
        motorR.setSpeed(map(InputWidth, signatures[i][1], setWidth, map(InputCoord, 165, 315, 255, 0),
0));
```

```
      }

      //if can/cup is to the left - R motor based solely off width, L motor based off width with max speed affected by mid/maxR x coord
      else if (InputCoord < 140) {
       motorL.setSpeed(map(InputWidth, signatures[i][1], setWidth, map(InputCoord, 0, 160, 0, 255), 0));
       motorR.setSpeed(map(InputWidth, signatures[i][1], setWidth, 255, 0));
      }

     }

    }

    //if the pixycam doesn't see the can, then keep turning left until it is detected
    else {
     while (pixy.ccc.numBlocks == 0) {
      motorL.setSpeed(-255);
      motorR.setSpeed(255);
      pixy.ccc.getBlocks();
     }
    }
   }

   //restart mode
   while (mode == 2) {
    brake();
   }

} //end of void loop()

//line following functions:

//detects all three ir sensors
void detect() {
 irStatusL = digitalRead(irSensorL);
 irStatusM = digitalRead(irSensorM);
 irStatusR = digitalRead(irSensorR);
}

//forward motor spins
```

```
void forward() {
 motorL.setSpeed(medL);
 motorR.setSpeed(medR);
}

//determines what type of turn it is by detecting the left and right sensor
void turnDetect() {
 motorL.setSpeed(detectL);
 motorR.setSpeed(detectR);

 while(irStatusL == 0 && irStatusR == 0) {
  irStatusL = digitalRead(irSensorL);
  irStatusR = digitalRead(irSensorR);
 }
}

//sets the motor speeds to 0
void brake() {
 motorL.setSpeed(0);
 motorR.setSpeed(0);
}

//if function is called, left turn
void leftTurn() {
 motorL.setSpeed(LTML);
 motorR.setSpeed(LTMR);

//while the rover isn't straight yet after the turn, keep sensing until it is straight
 while (irStatusM != 1) {
  irStatusM = digitalRead(irSensorM);
 }
}

//if function is called, right turn
void rightTurn() {
 motorL.setSpeed(RTML);
 motorR.setSpeed(RTMR);

//while the rover isn't straight yet after the turn, keep sensing until it is straight
 while (irStatusM != 1) {
  irStatusM = digitalRead(irSensorM);
```

```
 }
}

//servo functions
void grab(){
 myServo.write(85);   //50 to 85 wraps claw around can
}

//forward after grab function
void afterGrab(){
 motorL.setSpeed(255);
 motorR.setSpeed(255);
 delay(300);
 motorL.setSpeed(0);
 motorR.setSpeed(0);
 delay(1000);
 myServo.write(30);
}
```

Appendix D: References

1. "The History of Robots." *The Economic Times*,

   https://economictimes.indiatimes.com/the-history-of-robots/articleshow/3075438.cms?from=mdr. Accessed 3 March 2023.

2. Bellis, Mary. "Eduardo San Juan and the Lunar Rover That Changed the Space Program." *ThoughtCo*, ThoughtCo, 22 June 2019,

   https://www.thoughtco.com/eduardo-san-juan-and-moon-buggy-1991716#:~:text=Mechanical%20engineer%20Eduardo%20San%20Juan,designer%20of%20the%20Lunar%20Rover. Accessed 3 March 2023

3. "Mars Pathfinder." *NASA*, NASA, 7 Sept. 2019,

   https://mars.nasa.gov/mars-exploration/missions/pathfinder/#:~:text=Mars%20Pathfinder%20was%20launched%20December,surface%20of%20the%20red%20planet. Accessed 3 March 2023.

4. "Arduino Uno Rev3." *Arduino*. Accessed 23 March 2023.

   https://store-usa.arduino.cc/products/arduino-uno-rev3/

5. "Pixy2 CMUcam5." *Sparkfun*. Accessed 23 March 2023.

   https://www.sparkfun.com/products/14678

6. "34:1 Metal Gearmotor." *Pololu*. Accessed 23 March 2023.

   https://www.pololu.com/product/3204