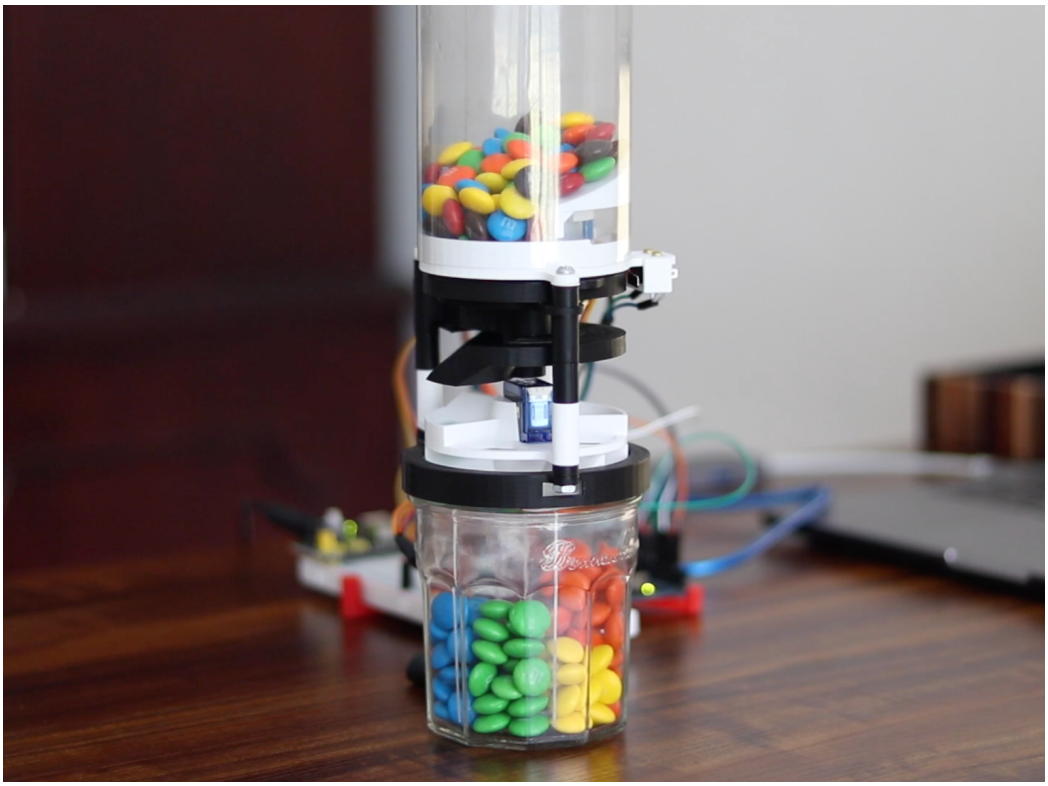
Arduino-Based M&M Sorting Machine



MAE 412 Final Project
Princeton University, Spring 2021
Jack Monaco

Abstract:
The prompt for this project was to use a limited assortment of sensors, electrical components, and motors to design a project that uses sensing, actuation, sequencing, and control to complete a task of our choice autonomously. This machine was designed to sort m&m candies by color. The system uses a combination of an RGB LED and a photoresistor to measure each m&m's response to several wavelengths of light. Code running on an Arduino Uno associates the response with a color, and a combination of mechanisms, driven by a stepper motor and a servo motor, sorts the m&m into one of six compartments.

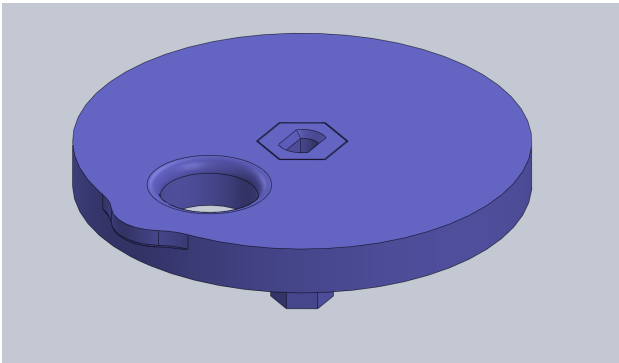
Design Description:

The central functions of this machine include singling out one m&m from an m&m reservoir, performing the sensing sequence on that m&m (using the photoresistor to measure the intensity of the reflection of red, green, and blue light), and sorting the m&m into a compartment based on the result. In order to use this sensing setup to collect reliably comparable photoresistor values, the lighting and positioning had to be very consistent for each m&m being tested.

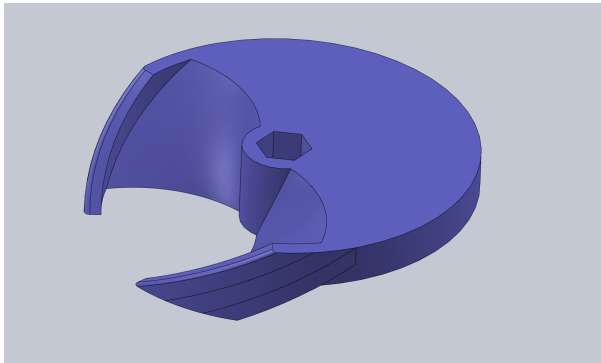
<u>Mechanism 1</u>

The first mechanism was designed to perform the majority of these tasks. The mechanism consists of two circular pieces (Parts 2 and 3) that lock together to rotate on the top and bottom surface of Part 1. The top disk (Part 2) has a hole to select one m&m from a reservoir above the mechanism, as well as a small cam. The bottom disk has a platform that drops off into two ramps. The pieces are oriented to one another as shown in the assembly.

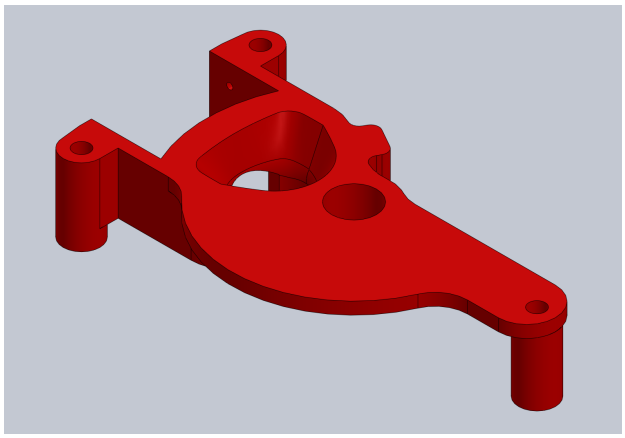
This system of two disks rotates within Part 1, which has the main chamber for m&m sensing. It is driven from above by a stepper motor mounted in Part 5. Part 5 also mounts a limit switch that rides up against the cam on the top disk.



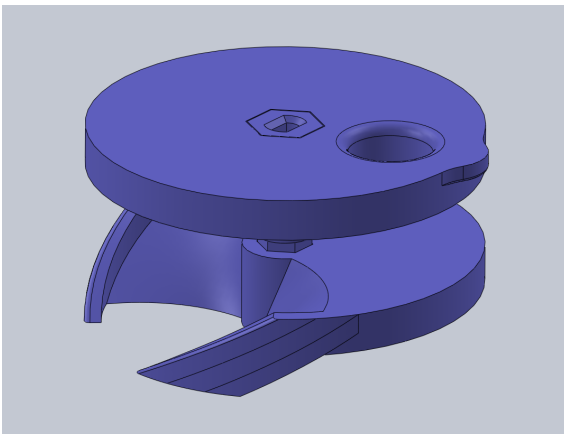
Part 2: m&m Selector Disk and Center Pin (can be printed as 1 or 2 parts)



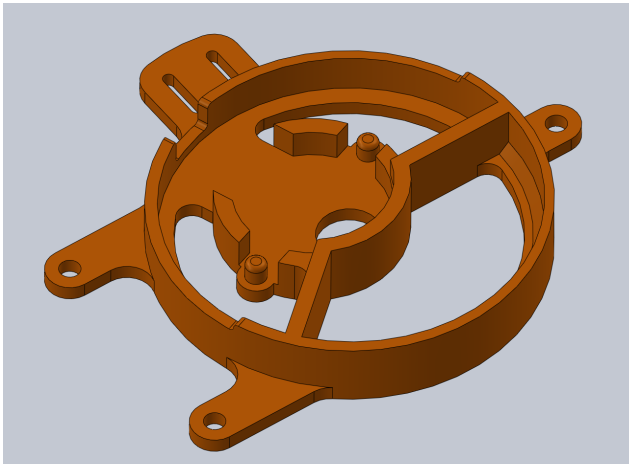
Part 3: Double Ramp Disk



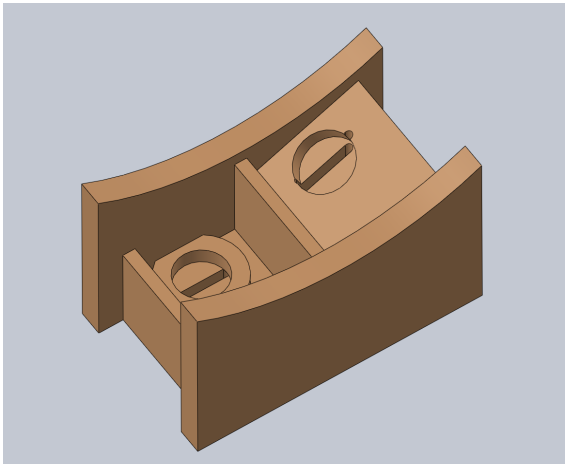
Part 1: Central Sensing Compartment



Rotator Assembly



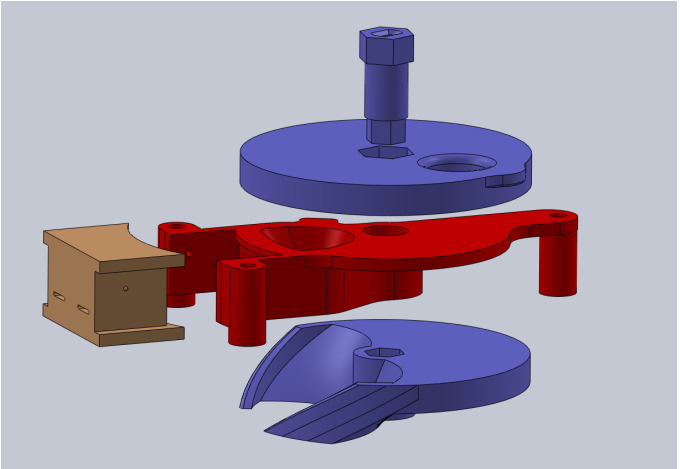
Part 5: Stepper Motor and Limit switch mount



Part 4: LED and Photoresistor housing

The rotation of Parts 1 and 2 performs several functions. First, the top disk selects an m&m from the reservoir, and by rotating 180 degrees clockwise drops it into the sensing chamber and closes the chamber on top and bottom. The chamber places the m&m in a controlled position where it can be viewed from the side. Part 4 snaps onto the side of Part 1 and houses the RGB LED and photoresistor. A white LED is also mounted into the back of the sensing chamber to, in conjunction with the photoresistor, detect if an m&m is in the chamber.

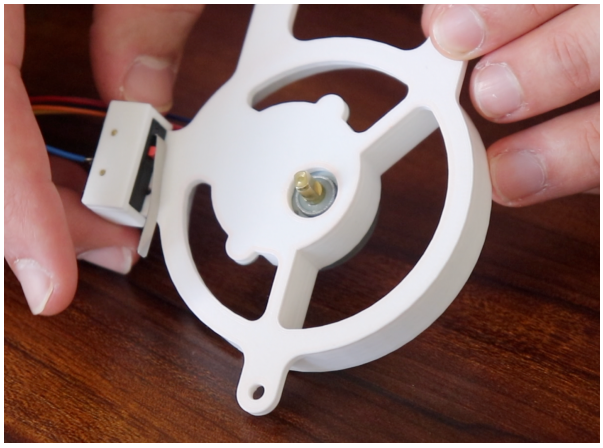
If an m&m has been successfully loaded, the RGB LED and photoresistor will collect values and then the mechanism will rotate 180 degrees to return to the loading position. Depending on the direction of rotation, either clockwise or counter clockwise, the m&m will meet one of the two ramps on the bottom disk, directing it either to the left or to the right. This constitutes the first sorting action.



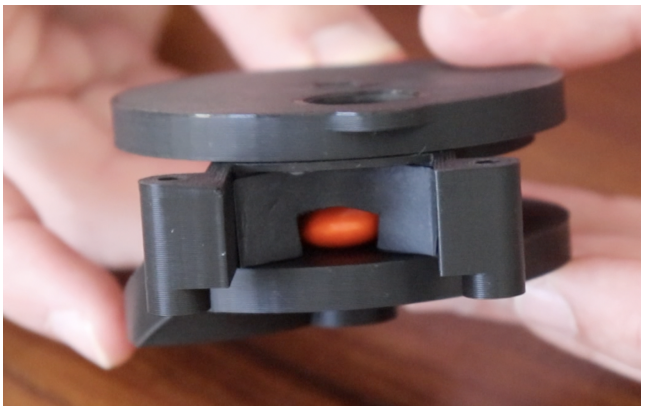
Parts 1-4 Assembly Order

Parts 1-3 Assembled

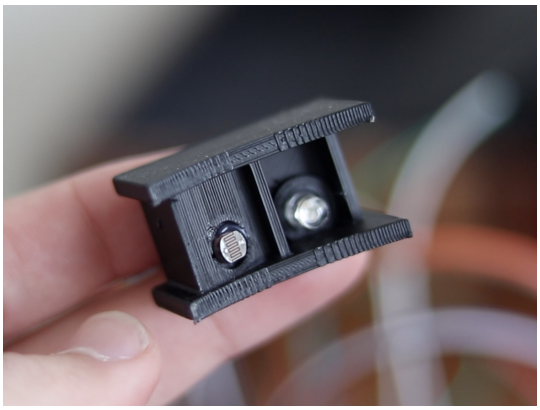
Stepper Motor Installed in Part 5



Stepper Motor and Limit switch in Part 5



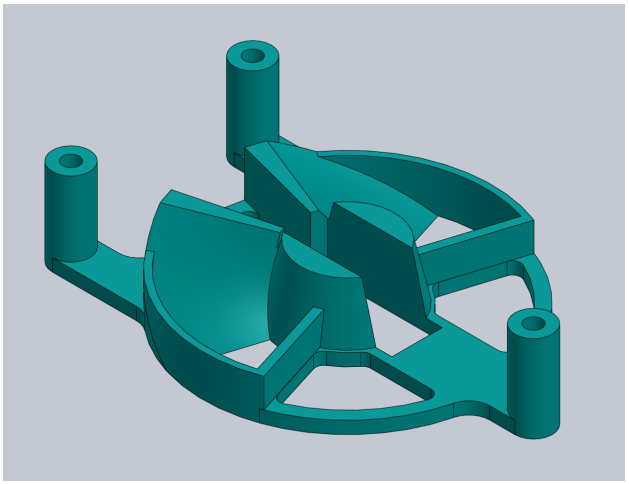
m&m loaded in sensing chamber, seen from side



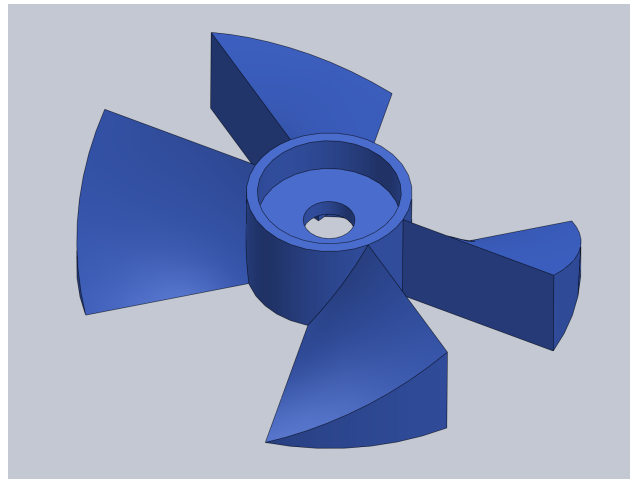
Part 4: LED and sensor housing

Mechanism 2

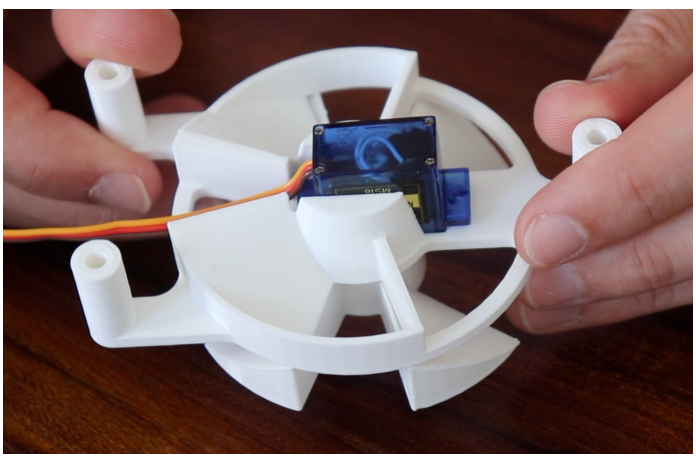
The second sorting mechanism is composed of Parts 7 and 8. The m&m dispensed from Mechanism 1 falls into the left or right side of Part 7, which also holds a servo motor which faces downward and rotates Part 8. Part 7 directs the m&m into one of two locations on opposite sides of the servo motor. Depending on the rotation of Part 8, the m&m then falls either straight down or encounters a ramp directing it 60 degrees clockwise or counterclockwise.



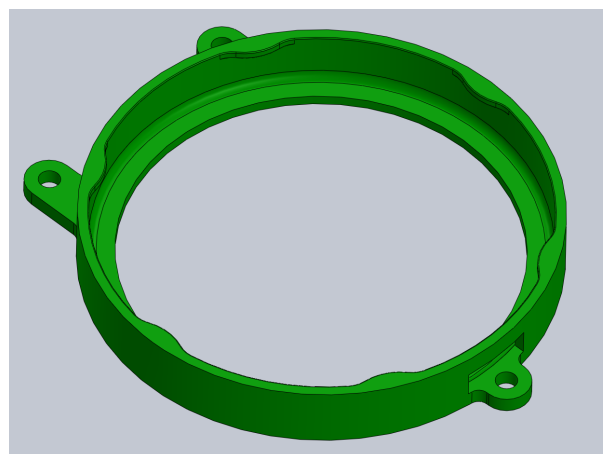
Part 7: 2-way funnel and servo mount



Part 8: 2-sided final divider



Parts 7 and 8 with servo installed



Part 9: Jar Lid

The entire sorter is mounted on top of Part 9, which is a lid that screws onto a mason jar outfitted with 6 clear dividers. With the two sorting directions of Mechanism 1 and the three sorting directions of Mechanism 2, the sorter can place the m&m in any of the six compartment colors.

Design Notes

My process for this design started with the concept for Mechanism 1, which I then constructed and tested. Most of the parts for this project took about 4-5 iterations to get them working exactly as I wanted. This was partially because as I was redesigning to include new features as I went. For instance, when using the stepper motor I was having trouble with the motor skipping steps or stalling at higher speeds. In a redesign I added the limit switch mount and cam so that I could zero the position of the rotator effectively and reliably. I also had issues with the loading system which led me to increase the area of the loading site and increase the angle of Part 6, which funnels the m&ms into the site.

After designing Mechanism 1, I played around with several ideas to multiply the number of colors the machine could sort. I ended up opting for the option that included the least number of moving parts, so as to increase robustness and simplicity. I was also limited by the selection of motors that I had available.
I was keeping my test m&ms in a small jam jar, and the parts for Mechanism 1 happened to fit very nicely into the mouth of the jar, so I ended up deciding to lean into that idea and model the whole machine to screw onto a jar. In keeping with the vertical design I modeled Parts 5 and 6 to fit within a clear water bottle that I had.

Since I had limited access to basic tools during this build, including a drill, I tried to utilize my one tool, the 3D printer, as much as possible. I did many test prints in order to be able to model my parts for minimal cleanup. These included screw holes and mounting holes for the stepper motor, servo motor, limit switch, and LEDs. I also iterated the pieces to fit the various physical components that I was working with, including the jam jar and water bottle.

I was able to troubleshoot most printing issues quickly by releveling the bed and cleaning the printer. I did encounter some issues with the bottom layer of a print melting a bit and flattening. This was especially problematic when my parts had precise hole diameters on the bottom layer. This is, for instance, why I ended up constructing Part 1 as two pieces, when it could have been printed as one.

All parts in SolidWorks and printed them on the Ender 3 Pro, mostly at 0.28mm layer height, with support as necessary. Parts 1-4 and printed in black PLA so as to keep the sensing chamber as dark as possible. I chose to print all the other parts in black and white PLA, purely for aesthetic reasons, so that the only color in the final machine was the color of the m&ms.

Full Assembly:

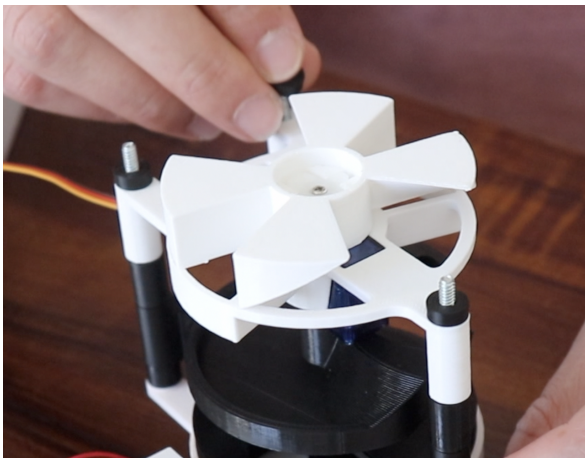
Mechanical

The Rotator (Parts 2 and 3) fits together snuggly and is assembled in Part 1 as shown. The lower disk is held in place with a small screw from the bottom. The stepper motor (in this case the 28BYJ-48) snaps into Part 5 facing downwards with the wires exiting through the slot. Printed spacers are used to offset each layer from the next as shown in the full assembly drawing. Because combinations of spacers of many lengths were used, only included one exemplary drawing has been included.

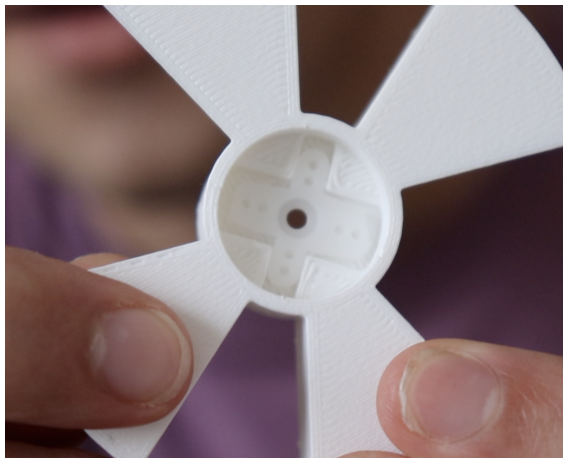
Part 6 fits snuggly onto the top of the stepper motor and directs m&ms in the reservoir towards the loading site. Part 10 fits around the limit switch to be held in place on the edge of Part 5 with two screws through the slots. The slots allow for the position of the limit switch to be adjusted. The RGB LED is placed into the deeper section of Part 4 and hot glued in place. The photoresistor is hot glued in place as well. Part 4 snaps onto the side of Part 1. A white LED is also hot glued into the LED housing in the back of the chamber in Part 1.

The servo motor (an SG90) fits snuggly into Part 7 facing down with the shaft in line with the center of the machine. Part 8 is designed such that one of the included servo motor attachment arms snaps into the piece, and a small screw from the bottom holds the attachment arm and Part 8 to the servo shaft.

The entire machine (Parts 1-10) are assembled using three, 3” 6-32 machine screws and corresponding nuts. Printed spacers separate each level the necessary amount (detailed in assembly drawings).



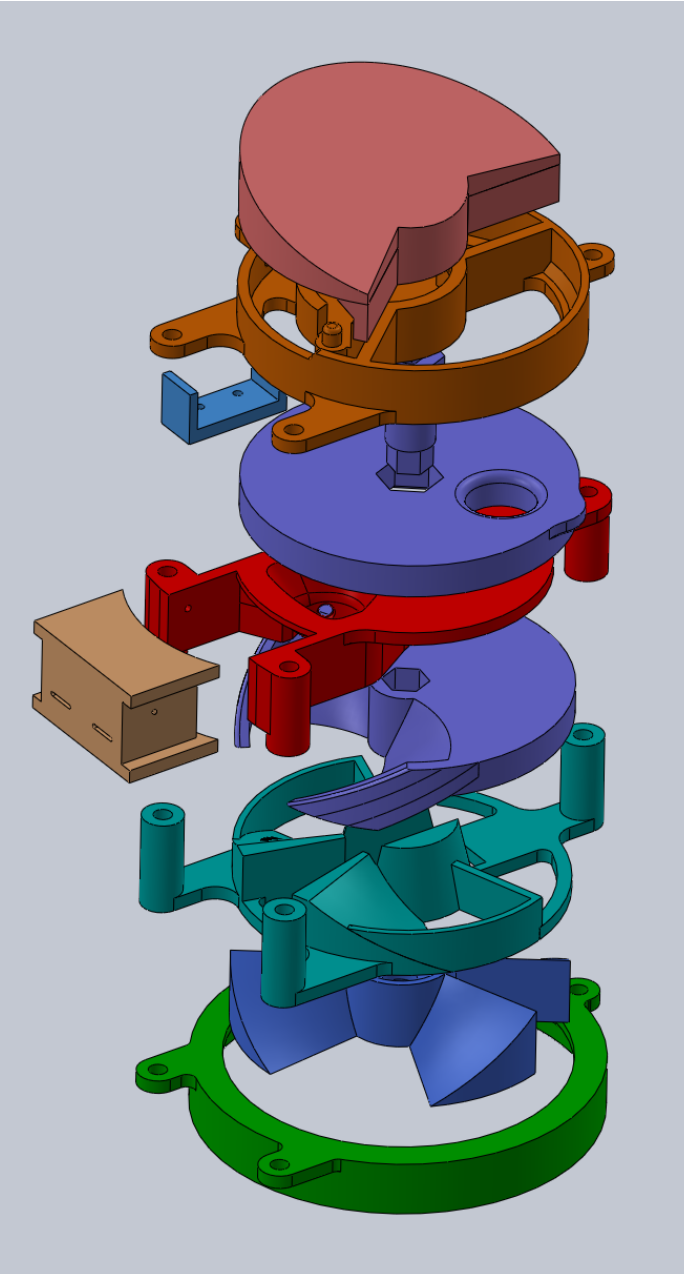
Adding Spacers to Assembly



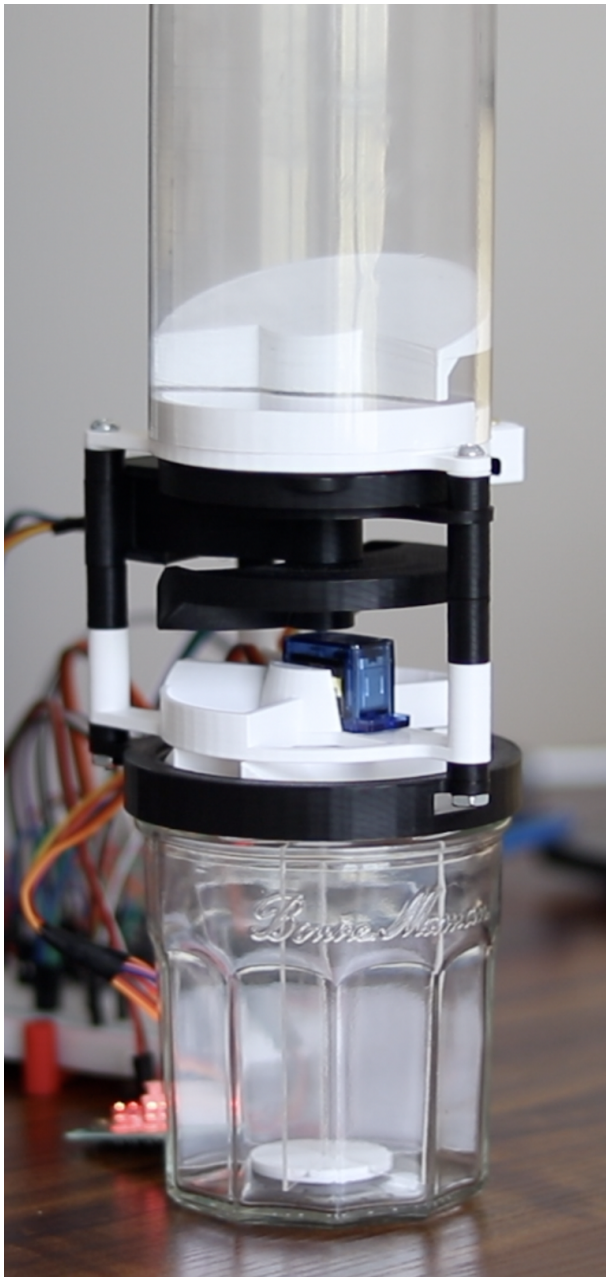
Servo Attachment in Part 8

The plastic water bottle (brand: SmartWater) is cut using a razor blade to fit onto Parts 5 and 6, with a hole cut in the top and a notch cut for the Stepper motor wires.

The dividers for the jam jar are made by scoring and snapping some 0.8mm plexiglass to make pieces that fit within the jar to divide it into 6 sections. These are held in place with Parts 11 and 12 (not included in below assembly).



All 3D Printed parts in order of Assembly

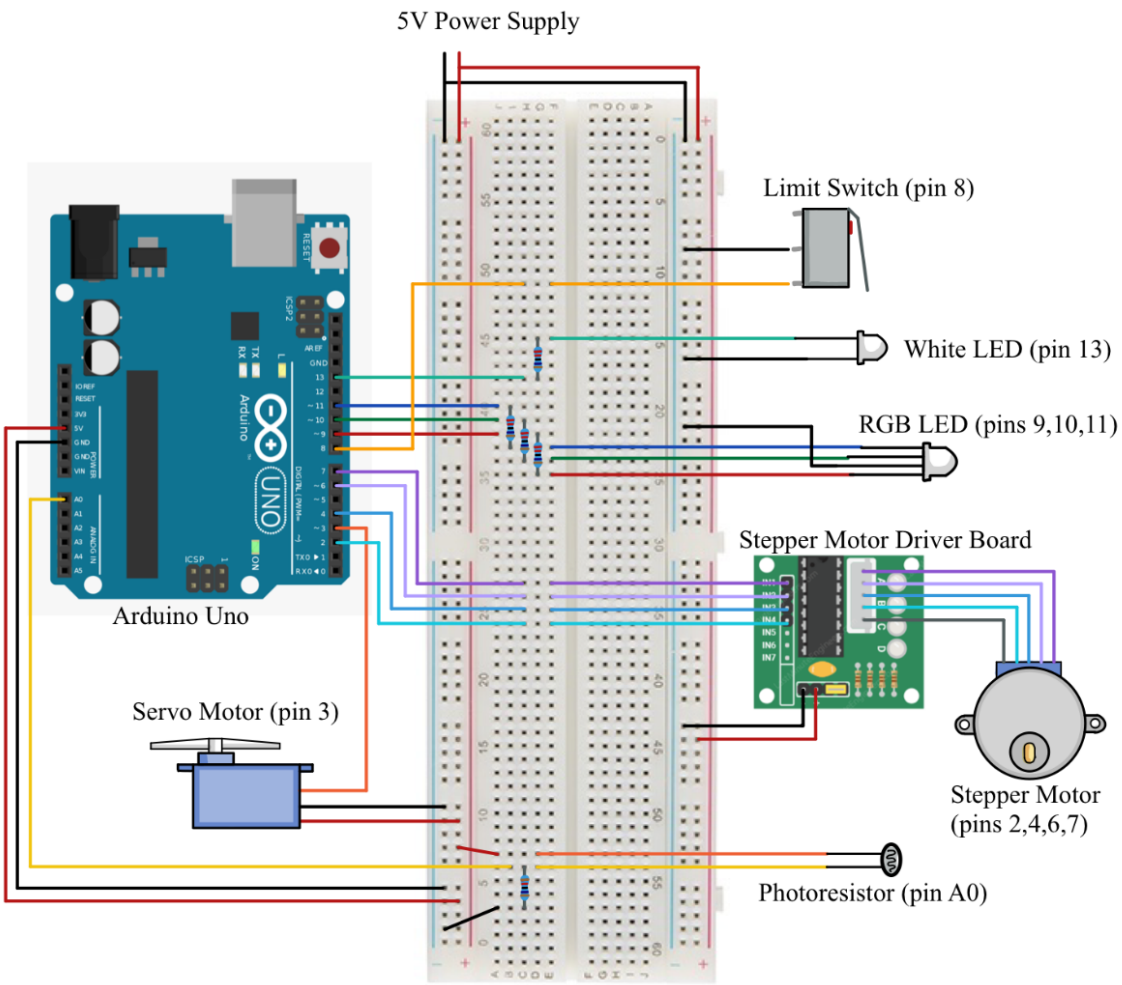


Fully Assembled Machine

Wiring

The machine is controlled by an Arduino Uno microcontroller. Pins 2,4,6, and 7 send digital signals to a stepper motor driver board (in this case the ULN2003) which also receives 5 volts of power. The servo motor is driven with PWM using pin 3. The RGB LED is also controlled using PWM from pins 9, 10, and 11, and each LED, including the white LED in the back of the chamber, is wired with a 220 Ohm resistor. The limit switch is wired to pin 13 and uses the Arduino's internal pullup resistor.

The final electrical component is the photoresistor, which is wired in a voltage divider with a 10 KiloOhm resistor. Analog pin A0 reads the intermediate voltage from the voltage divider.



Wiring Diagram

Coding:

Full code can be seen in the Appendix. Below is a description of each section.

Initialization/Setup

This code only runs once. In this section pins are assigned to input/output, libraries to run the stepper motor and servo motor are loaded, and both the stepper and servo objects are created and initialized.

A boolean variable SetUp is set to = false.

Also in this section seven arrays are created, six of which will serve to store the reference values for the six m&m colors, and one of which will be reused for each m&m test. Each array holds three photoresistor readings and a tally for how many m&ms that array has identified.

Functions

Brief overview of each function and when it is used.

MovetoLimit()

This function moves the stepper motor clockwise until the limit switch is hit. It also uses a basic for loop to time how long the stepper has been moving. If too much time has passed, that means the top disk has gotten jammed, in which case the function will briefly reverse the stepper direction and then continue clockwise. This does an excellent job of catching mechanical jams. Function is called by *Sort()* and *SetupSort()*.

setLED()

Function sets the RGB LED to a set of three PWM values. Function is called by *CollectColor()*.

CollectColor(int samples, int ColorArray[])

Function takes a color array and int *samples* as arguments. When called, function flashes the white LED and takes a photoresistor value to determine if the chamber is loaded. If so, the function shines red, blue, and green light from the RGB LED, takes corresponding photoresistor values, and stores them in the array. With *samples* > 1, each photoresistor value is an average of (*samples*) readings. This feature may not have a significant effect on accuracy. If no m&m is detected, Array is set to zeros. Function is called by *CollectReferences()*.

Sort(int ID)

Function controls servo motor and stepper motor to dispense m&m into one of compartments 1 through 6, depending on ID. If fed zero, stepper continues clockwise to load another m&m.

SetupSort(int ID)
Has the exact same function as *Sort()* but with slightly different rotation values calibrated for the expectation that there is only one m&m in the loading site. Only called during *CollectReferences()*

CollectReferences(int Color1[] … int Color6[])
Function takes all reference arrays and, using *SetupSort()* calls *CollectColor()* to collect initial references for the six colors, based off of the first six m&ms it sees.

FindMatch(int TestColor[], int Color1[] … int Color6[])
Function takes the test array and compares it to the reference values in the six reference arrays using a least squares approach. For the closest array, the function returns an int with the corresponding ID, if no m&m was detected (ie. TestArray = [0,0,0,..]), function returns zero. *FindMatch* also calls *UpdateReference()* with the ID of the identified color.

UpdateReference(TestArray[], ColorArray[])
Function takes the test array and the color reference of the identified color. Then, using the tally for that color (the fourth value of the reference array) it updates the reference values for that color to be an average of all identified m&ms of that color, including the one just identified (TestArray[]). This feature was added to correct for some occasional misidentifications and has increased robustness dramatically. Function is called by *FindMatch()*.

EndScript(int match, int VoidTally, int TestArray[], int Color1[] … int Color6[])
This function keeps track using the variable VoidTally of how many consecutive times the machine has failed to load an m&m into the chamber. If that number reaches 10, it is assumed that the m&m reservoir is empty and the function prints the statistics for the run and ends the program using exit(0);

<u>Main Loop</u>
The main loop starts by checking if boolean SetUp == false. If so, it calls *CollectReferences()* to collect initial reference reference values and sets SetUp = true.

With an m&m now loaded, the loop calls *CollectColor()* to sense the loaded m&m.
Then, *FindMatch()* identifies the m&m and updates the associated reference values.
Using *Sort()* the program sorts the m&m and simultaneously loads another.

The identification ID is fed to *EndScript()* so that if the chamber has been empty ten times in a row, the program will terminate.

Results and Future Improvements:

This machine is a very successful proof of concept.

For those curious, testing for this project has suggested that blue and orange are the most common m&m colors, showing up with nearly twice the frequency of the other colors. This is consistent with published data from the Mars New Jersey Factory.

A full run of sorting can fit about 150 m&ms. This design is definitely not the most efficient, but it can sort around 10 m&ms per minute, and can be left alone while doing so. It does a great job of catching jams using the *MovetoLimit* function. When the reservoir is fully loaded, the m&ms have a harder time loading because of the increased vertical pressure, and occasionally this will mean the mechanism rotates several times before successfully loading. In testing this has only once come close to accidentally triggering the *EndScript* function, but in the future this could be corrected using an improved feeder system, or even an auger in the reservoir.

Most importantly, the color sensing system now works flawlessly. Since implementing the method to update the reference values, no m&ms have been misidentified. The only error that does occasionally happen has to do with Part 8, the final rotating piece. Part 8 is modeled to be compact, and in retrospect it is a bit too shallow. When rotated to direct the m&m clockwise or counterclockwise, the m&m does not have a lot of space to slide through and so occasionally it will get stuck and not slide through Part 8. Then, when the servo returns to the middle position, that m&m will fall through the center channel and be missorted. This can sometimes be corrected by adjusting how far Part 8 rotates, but this poses additional problems, and in a run of 150 m&ms, there are still typically 1-3 missorts.

In a future version, Part 8 could be redesigned to fix this problem. This improvement was not made in this version due to time constraint and a limitation on the height of the machine given the bolts being used. In the future the wiring could be moved off of the breadboard and onto some type of circuit board or Arduino shield so that the whole contraption could be mounted on the machine and battery powered. A future version of this project could also tackle the important task of sorting Skittles.

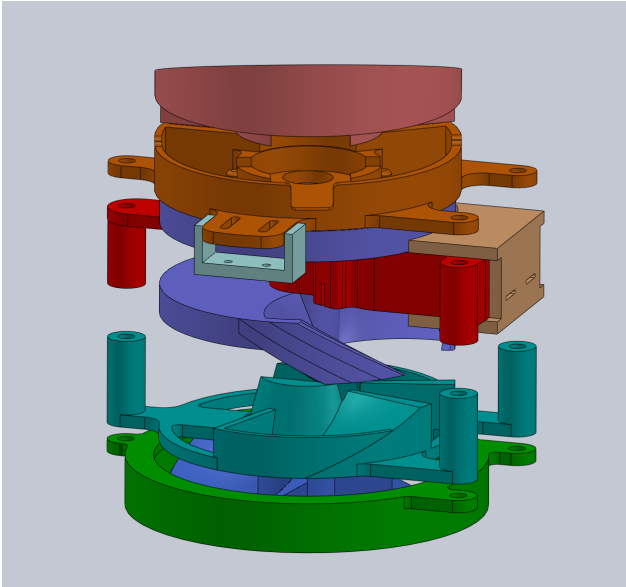
For a video of the machine working, see www.jackmonaco.design

Appendix:

Additional Images

Machine Sorting Progression at 1 minute, 7 minutes, 15 minutes

Dividers held with Parts 11 and 12



Collapsed Assembly

White LED install into the back of the sensing chamber

Arduino Code

```
/////INITIALIZE//////////////////////////////////////////////////////////////////////////////////////////////////////////////
  int redPin = 9;
  int greenPin = 10;
  int bluePin = 11;
  int whitePin = 13;
  int limitPin = 8;
  int servoPin = 3;

  int Color1[] = {0,0,0,0};
  int Color2[] = {0,0,0,0};
  int Color3[] = {0,0,0,0};
  int Color4[] = {0,0,0,0};
  int Color5[] = {0,0,0,0};
  int Color6[] = {0,0,0,0};

  int TestColor[] = {0,0,0,6};
  int VoidTally = 0;
  boolean SetUp = false;

#include <Stepper.h>
Stepper myStepper = Stepper(2048,2,6,4,7);

#include <Servo.h>
Servo myServo;


/////SETUP//////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void setup() {
  // set LEDs to output and limit switch to input w/ pullup resistor
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
```

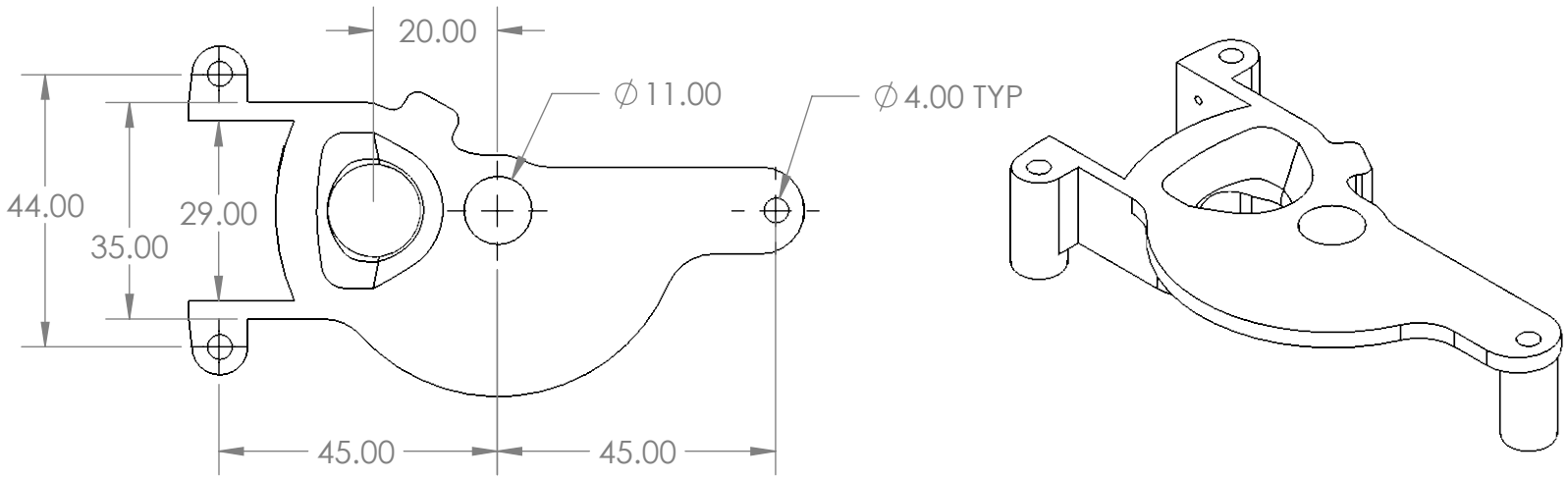
```
  pinMode(whitePin, OUTPUT);
  pinMode(limitPin, INPUT_PULLUP);

  //Stepper motor, Servo motor, and serial port
  myStepper.setSpeed(15);
  myServo.attach(servoPin);
  myServo.write(80);
  Serial.begin(9600);
}

/////MAIN////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void loop() {

 // startup routine for first 6 m&ms
  if(SetUp == false){
    CollectReferences(Color1,Color2,Color3,Color4,Color5,Color6);
    SetUp = true;
  }

  // collect value for color being tested
  CollectColor(5, TestColor);

  // math to determine closest one
  int match = FindMatch(TestColor,Color1,Color2,Color3,Color4,Color5,Color6);

  // sorts based on result
  Sort(match);

  // ends sequence and prints stats if there have been 10 empty rounds in a row
  VoidTally = EndScript(match, VoidTally,TestColor,Color1,Color2,Color3,Color4,Color5,Color6);

}

/////FUNCTIONS//////////////////////////////////////////////////////////////////////////////////////////////////////////////

void CollectReferences(int Color1[],int Color2[],int Color3[],int Color4[],int Color5[],int Color6[]){

  //stepper move to start position
  MovetoLimit();
  delay(1000);

  //loads and stores data stepping through colors... trust me
  myStepper.step(1400); //LOAD 1
  MovetoLimit();
  CollectColor(10,Color1);

  SetupSort(1); //SORT 1, LOAD 2
  CollectColor(10,Color2);

  SetupSort(2); //SORT 2, LOAD 3
  CollectColor(10,Color3);

  SetupSort(3); //SORT 3, LOAD 4
  CollectColor(10,Color4);

  SetupSort(4); //SORT 4, LOAD 5
```

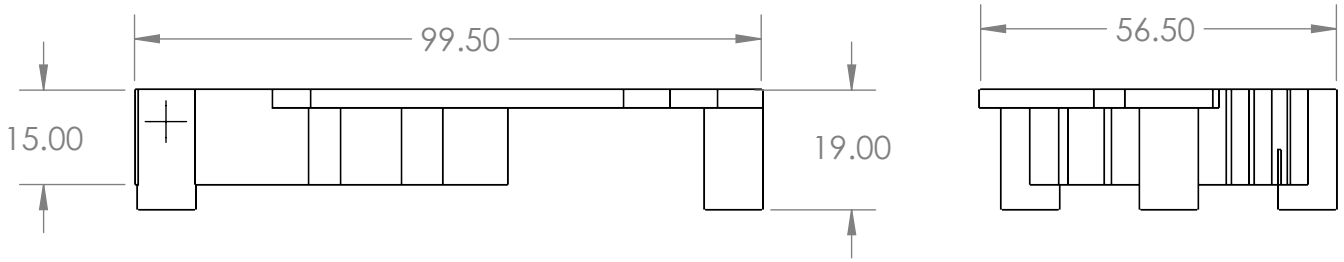
```
  CollectColor(10,Color5);

  SetupSort(5); //SORT 5, LOAD 6
  CollectColor(10,Color6);

  SetupSort(6); //SORT 6
}


void SetupSort(int ID){
  // sorting function for first 6 m&ms
  int m = 80;
  int h = 43;
  switch(ID){
    case 1: myServo.write(m+h); myStepper.step(1400); myServo.write(m); MovetoLimit();
    break;
    case 2: myStepper.step(1400); MovetoLimit();
    break;
    case 3: myServo.write(m-h); myStepper.step(1400); myServo.write(m); MovetoLimit();
    break;
    case 4: myServo.write(m+h); MovetoLimit(); myServo.write(m);
    break;
    case 5: MovetoLimit();
    break;
    case 6: myServo.write(m-h); MovetoLimit(); myServo.write(m);
    break;
  }
}

void Sort(int ID){
  // Sorting function controls dispensing sequence
  int m = 80;
  int h = 43;
  switch(ID){
    case 0: MovetoLimit();
    break;
    case 1: myServo.write(m+h); myStepper.step(1050); myServo.write(m); MovetoLimit();
    break;
    case 2: myStepper.step(1050); MovetoLimit();
    break;
    case 3: myServo.write(m-h); myStepper.step(1050); myServo.write(m); MovetoLimit();
    break;
    case 4: myServo.write(m+h); MovetoLimit(); myServo.write(m);
    break;
    case 5: MovetoLimit();
    break;
    case 6: myServo.write(m-h); MovetoLimit(); myServo.write(m);
    break;
  }
}


void MovetoLimit(){
  // moves clockwise until limit switch is hit
  // i provides failsafe for jamming
  int i = 0;
```

```
  myStepper.step(-200);
  while(digitalRead(limitPin) == HIGH){
    myStepper.step(-10);
    if(i>200){
      myStepper.step(300);
      i = 0;
    }
    i++;
  }
}


void setLED(int redValue, int greenValue, int blueValue){
  // function writes three PWM values to RGB LED
  analogWrite(redPin, redValue);
  analogWrite(greenPin, greenValue);
  analogWrite(bluePin, blueValue);
}


void CollectColor(int samples, int ColorArray[]){

  //if the chamber is empty, return zeros
  digitalWrite(whitePin, HIGH);
  delay(50);
  int check = analogRead(A0);
  digitalWrite(whitePin, LOW);

  if(check>15){
    ColorArray[0] = 0;
    ColorArray[1] = 0;
    ColorArray[2] = 0;
    return;
  }
  delay(300);

  //collects photoresistor value from red, green, then blue light and stores average values in array
  setLED(255,0,0);
  int sum = 0;
  delay(200);
  for(int i=0;i<samples;i++){
    int red = analogRead(A0);
    sum = sum + red;
  }
  ColorArray[0] = sum/samples;

  setLED(0,255,0);
  sum = 0;
  delay(200);
  for(int i=0;i<samples;i++){
    int red = analogRead(A0);
    sum = sum + red;
  }
  ColorArray[1] = sum/samples;

  setLED(0,0,255);
```

```
  sum = 0;
  delay(200);
  for(int i=0;i<samples;i++){
    int red = analogRead(A0);
    sum = sum + red;
  }
  ColorArray[2] = sum/samples;

  // turn LED off and update overall m&m counter
  setLED(0,0,0);
  ColorArray[3]++; //this updates color array counters only in the setup function
                   //during the main loop this only updates overall counter

//  Serial.print(int(ColorArray[0]));
//  Serial.print("   ");
//  Serial.print(int(ColorArray[1]));
//  Serial.print("   ");
//  Serial.println(int(ColorArray[2]));
}


int FindMatch(int TestColor[],int Color1[],int Color2[],int Color3[],int Color4[],int Color5[],int Color6[]){
  // finds array of values closest to TestColor by least squares
  // also calls UpdateReference() to update the reference values, once match has been identified

  int match = 0;

  // if compartment is empty return 0
  if(TestColor[0] == 0){
    return match;
  }

  long sums[] = {0,0,0,0,0,0,0};
  for(int i=0;i<3;i++){
    sums[1] = sums[1] + pow(TestColor[i]-Color1[i],2);
    sums[2] = sums[2] + pow(TestColor[i]-Color2[i],2);
    sums[3] = sums[3] + pow(TestColor[i]-Color3[i],2);
    sums[4] = sums[4] + pow(TestColor[i]-Color4[i],2);
    sums[5] = sums[5] + pow(TestColor[i]-Color5[i],2);
    sums[6] = sums[6] + pow(TestColor[i]-Color6[i],2);
  }

  long closest = min(sums[1],min(sums[2],min(sums[3],min(sums[4],min(sums[5],sums[6])))));

  if(closest==sums[1]){ match = 1;
  UpdateReference(TestColor, Color1);
  }
  else if(closest==sums[2]){ match = 2;
  UpdateReference(TestColor, Color2);
  }
  else if(closest==sums[3]){ match = 3;
  UpdateReference(TestColor, Color3);
  }
  else if(closest==sums[4]){ match = 4;
  UpdateReference(TestColor, Color4);
  }
```

```
  else if(closest==sums[5]){ match = 5;
  UpdateReference(TestColor, Color5);
  }
  else { match = 6;
  UpdateReference(TestColor, Color6);
  }
  return match;
}


void UpdateReference(int TestArray[], int ColorArray[]){
  //updates the color reference array with newly associated values
  for(int i=0;i<3;i++){
    ColorArray[i] = ColorArray[i] + ((TestArray[i]-ColorArray[i])/(ColorArray[3]+1));
  }
  //increments the color array counter (how many of this color have been sorted)
  ColorArray[3]++;
}


int EndScript(int match,int VoidTally,int TestColor[],int Color1[],int Color2[],int Color3[],int Color4[],int
Color5[],int Color6[]){
  if(match == 0){ VoidTally++;
  }
  else{ VoidTally = 0;
  }

  if(VoidTally >= 10){
    Serial.println("Sorting Complete!");
    Serial.print("Total: ");
    Serial.println(TestColor[3]);
    Serial.print("Red: ");
    Serial.println(Color1[3]);
    Serial.print("Orange: ");
    Serial.println(Color2[3]);
    Serial.print("Yellow: ");
    Serial.println(Color3[3]);
    Serial.print("Green: ");
    Serial.println(Color4[3]);
    Serial.print("Blue: ");
    Serial.println(Color5[3]);
    Serial.print("Brown: ");
    Serial.println(Color6[3]);

    // ends program
    delay(1000);
    exit(0);
  }
  return VoidTally;
}
```





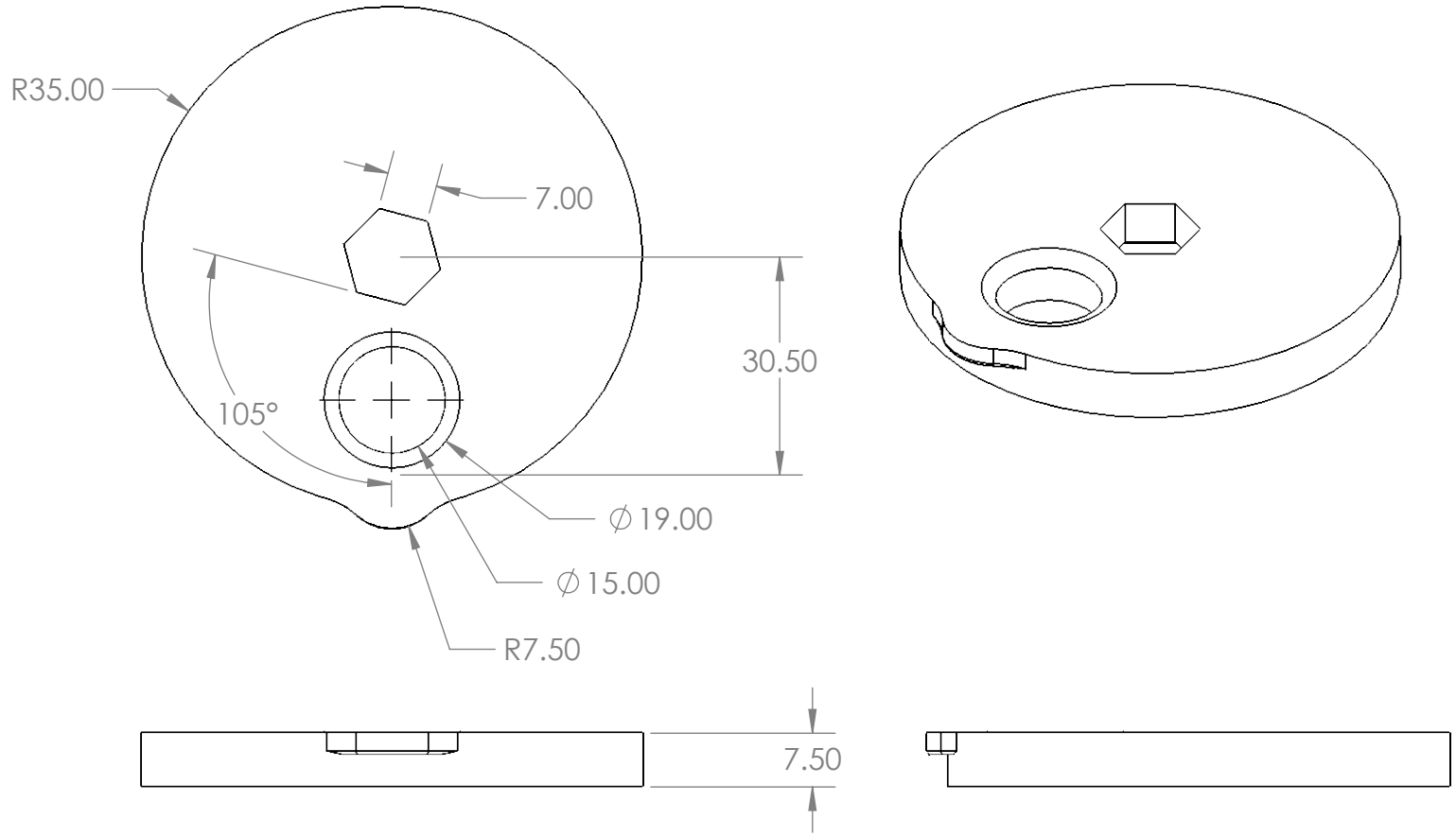
UNLESS OTHERWISE SPECIFIED:		NAME	DATE
DIMENSIONS ARE IN MILLIMETERS	DRAWN		
	CHECKED		
	ENG APPR.		
	MFG APPR.		
INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
MATERIAL PLA PLASTIC	COMMENTS: 3D PRINTED, NO CLEANUP NECESSARY		
FINISH PRINT FINISH			
DO NOT SCALE DRAWING			

TITLE:

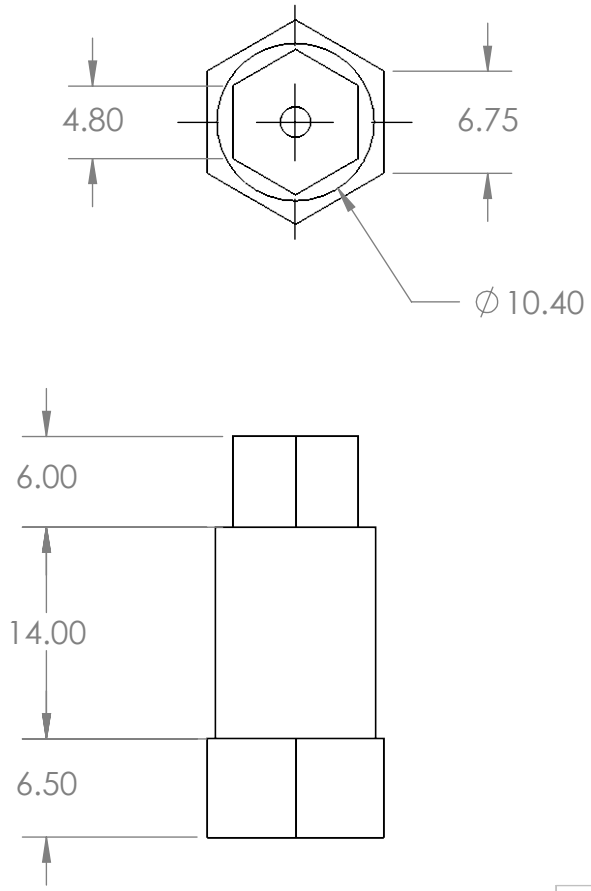
PART 1

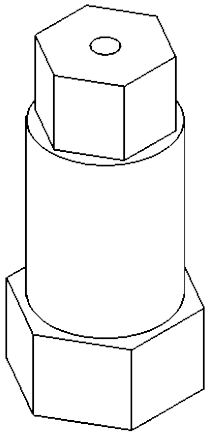
SIZE	DWG. NO.	REV
A		

SCALE: 1:2



UNLESS OTHERWISE SPECIFIED:		NAME	DATE	
DIMENSIONS ARE IN MILLIMETERS	DRAWN			
	CHECKED			TITLE: PART 2a
	ENG APPR.			
	MFG APPR.			
INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.			
MATERIAL PLA PLASTIC	COMMENTS: 3D PRINTED, NO CLEANUP NECESSARY			SIZE A / DWG. NO. / REV
FINISH PRINT FINISH				
DO NOT SCALE DRAWING				SCALE: 1:1





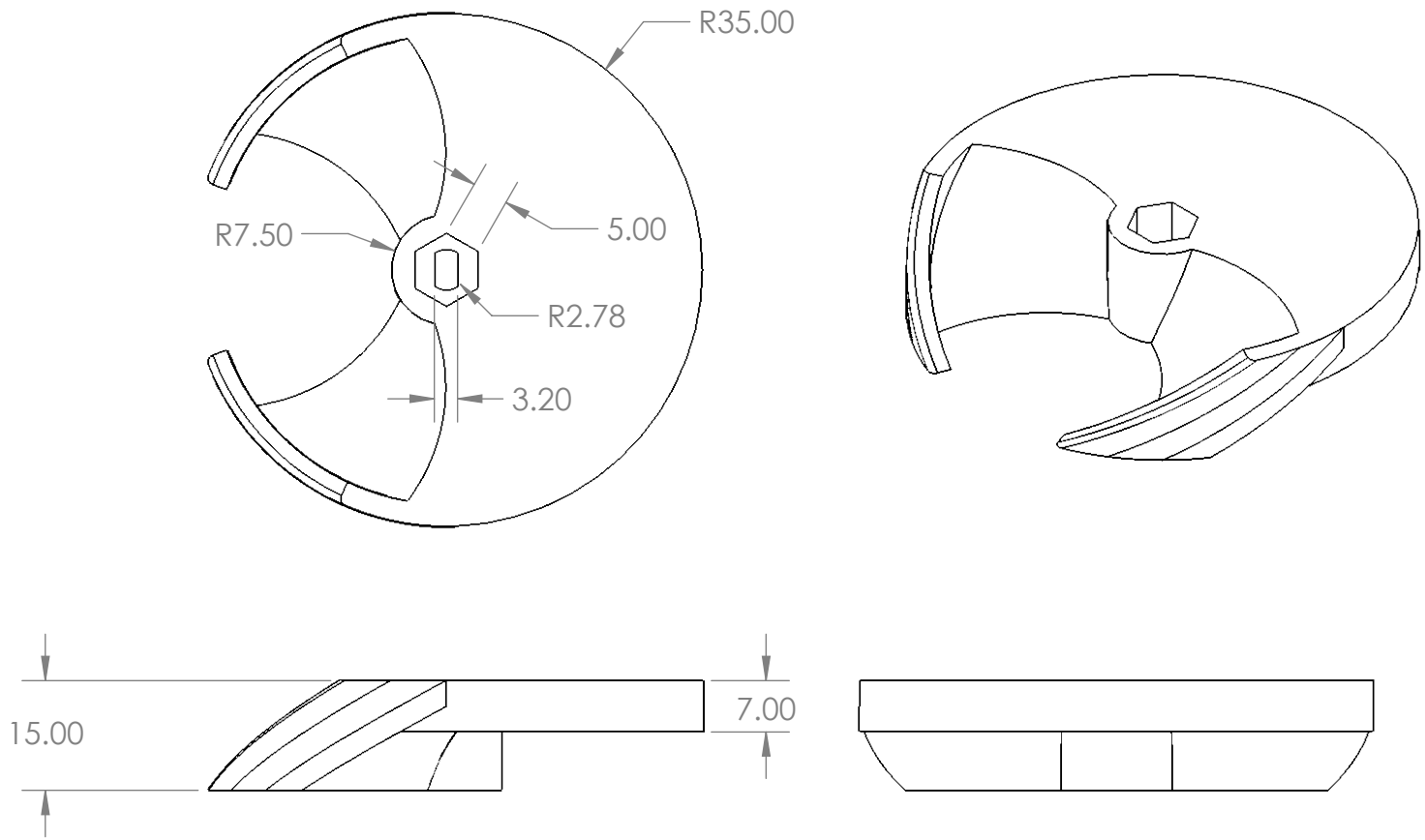
UNLESS OTHERWISE SPECIFIED:		NAME	DATE
DIMENSIONS ARE IN MILLIMETERS	DRAWN		
	CHECKED		
	ENG APPR.		
	MFG APPR.		
INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
MATERIAL PLA PLASTIC	COMMENTS: 3D PRINTED, NO CLEANUP NECESSARY		
FINISH PRINT FINISH			
DO NOT SCALE DRAWING			

TITLE:

PART 2b

SIZE	DWG. NO.	REV
A		

SCALE: 2:1

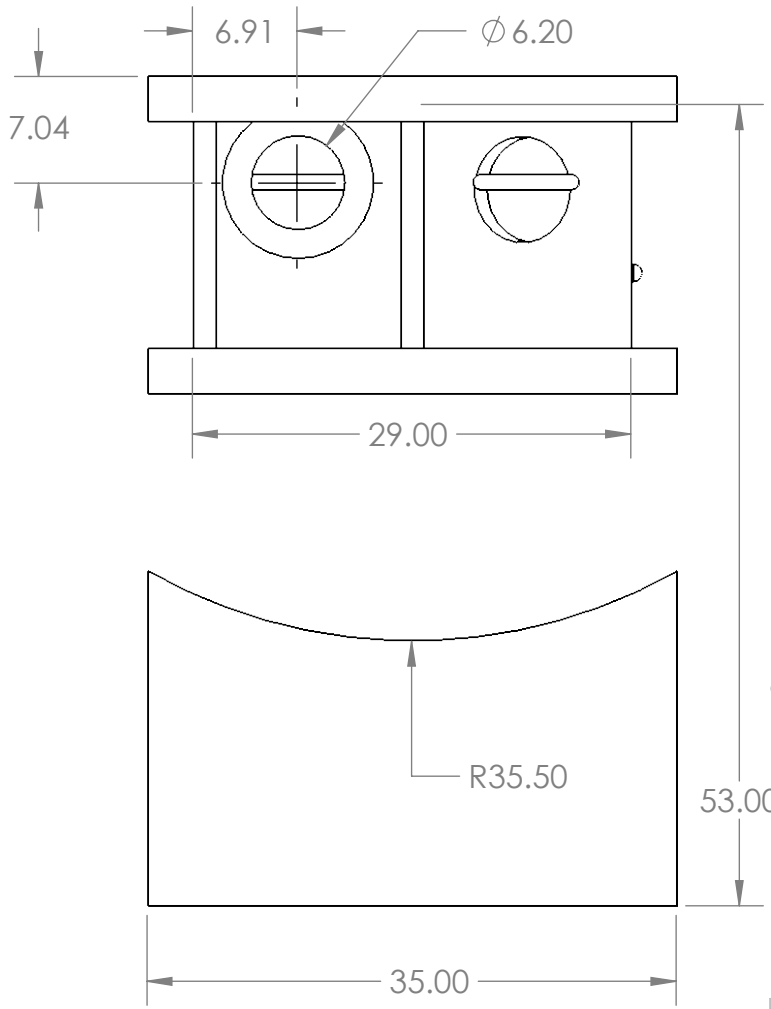


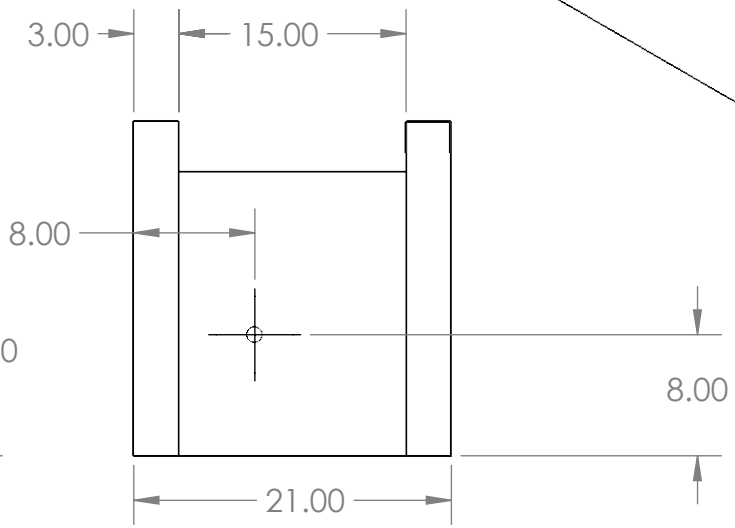
UNLESS OTHERWISE SPECIFIED:		NAME	DATE
DIMENSIONS ARE IN MILLIMETERS	DRAWN		
	CHECKED		
	ENG APPR.		
	MFG APPR.		
INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
MATERIAL PLA PLASTIC	COMMENTS: 3D PRINTED, NO CLEANUP NECESSARY		
FINISH PRINT FINISH			
DO NOT SCALE DRAWING			

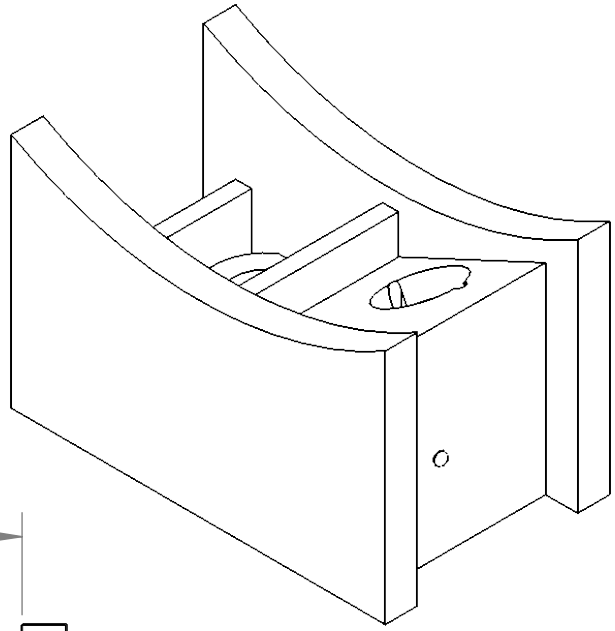
TITLE: PART 3

SIZE	DWG. NO.	REV
A		

SCALE: 1:1







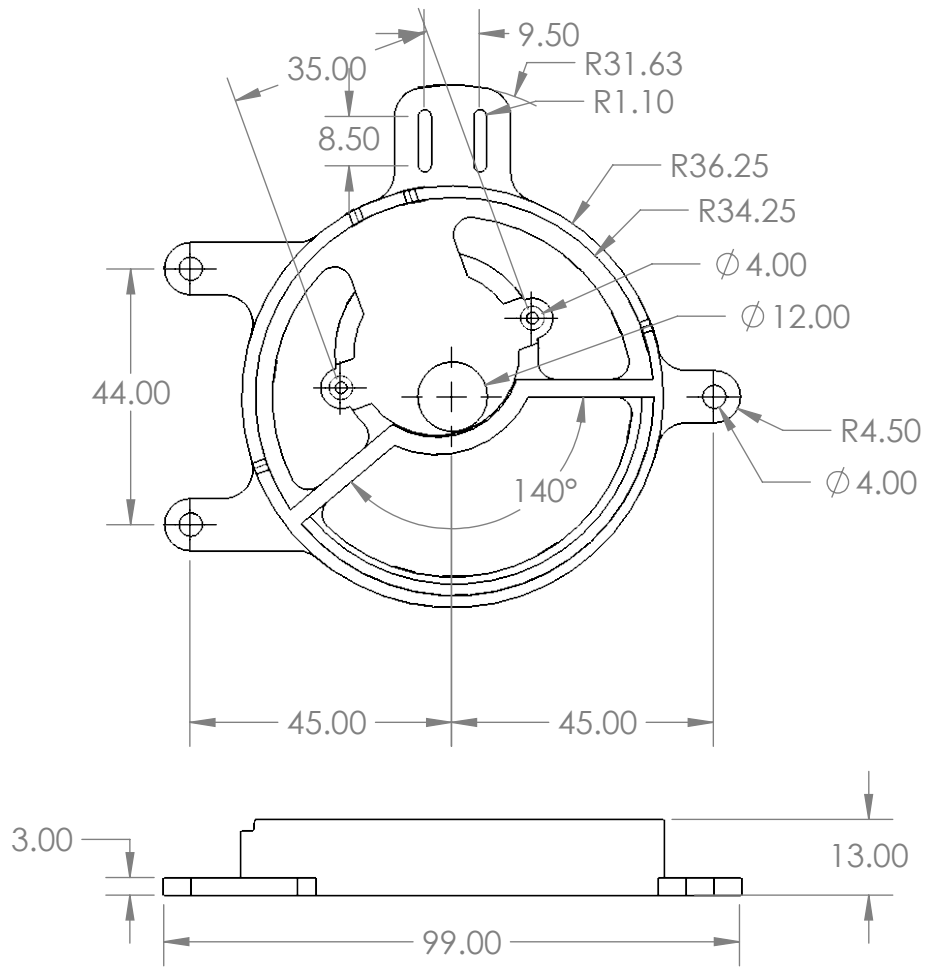
UNLESS OTHERWISE SPECIFIED:		NAME	DATE
DIMENSIONS ARE IN MILLIMETERS	DRAWN		
	CHECKED		
	ENG APPR.		
	MFG APPR.		
INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
MATERIAL PLA PLASTIC	COMMENTS: 3D PRINTED, NO CLEANUP NECESSARY		
FINISH PRINT FINISH			
DO NOT SCALE DRAWING			

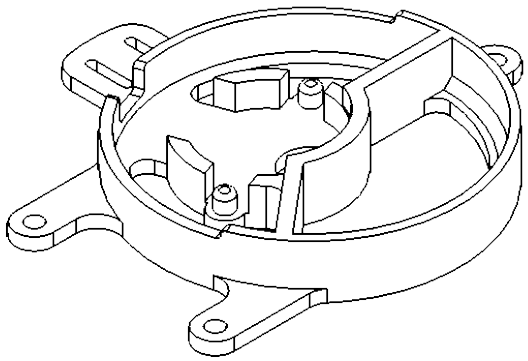
TITLE:

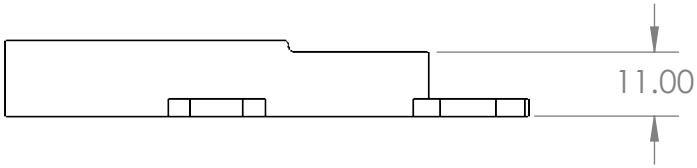
PART 4

SIZE	DWG. NO.	REV
A		

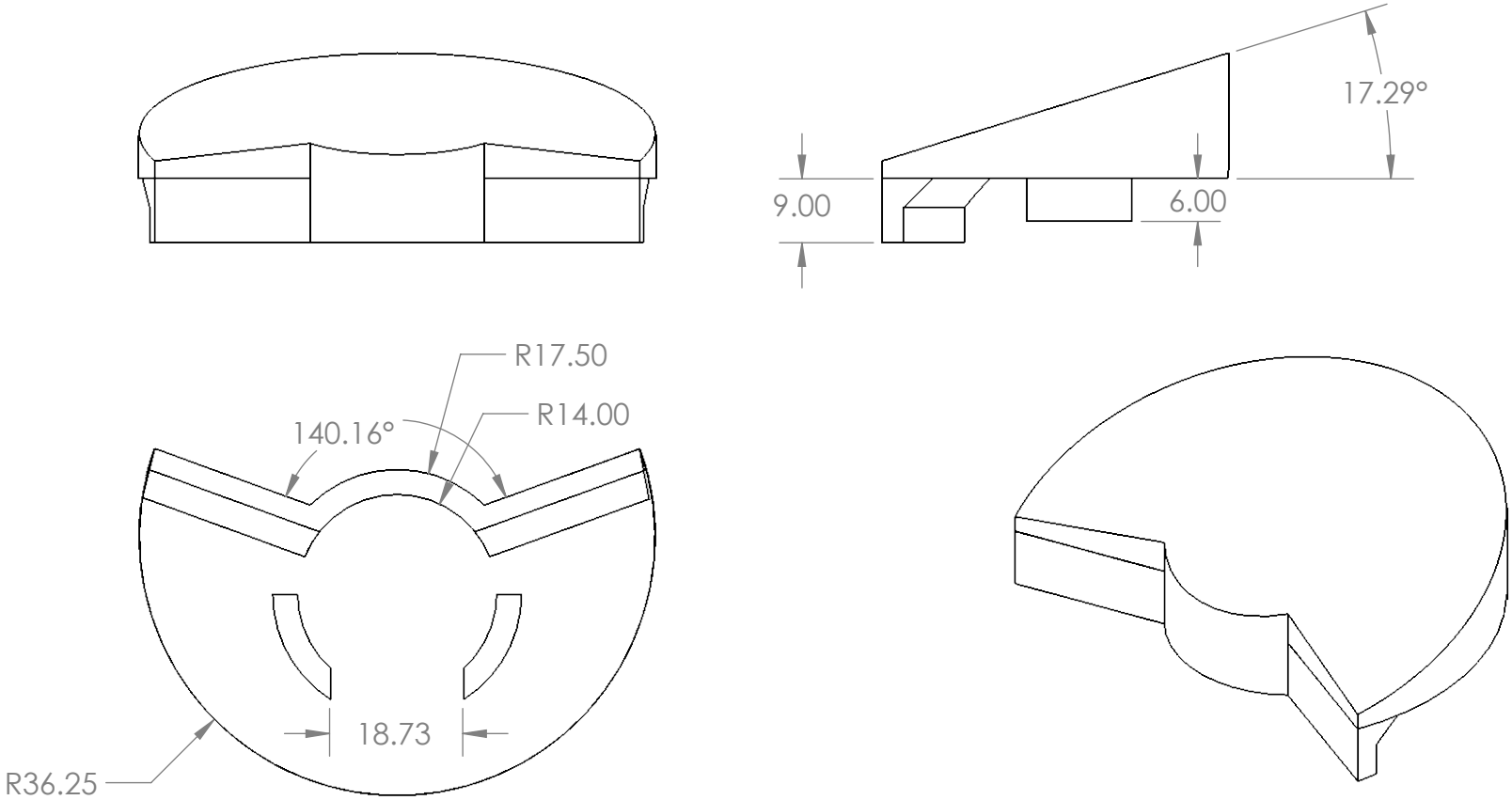
SCALE: 2:1







UNLESS OTHERWISE SPECIFIED:		NAME	DATE	
DIMENSIONS ARE IN MILLIMETERS	DRAWN			
	CHECKED			TITLE: PART 5
	ENG APPR.			
	MFG APPR.			
INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.			
MATERIAL PLA PLASTIC	COMMENTS: 3D PRINTED, NO CLEANUP NECESSARY			SIZE A / DWG. NO. / REV
FINISH PRINT FINISH				
DO NOT SCALE DRAWING				SCALE: 1:2



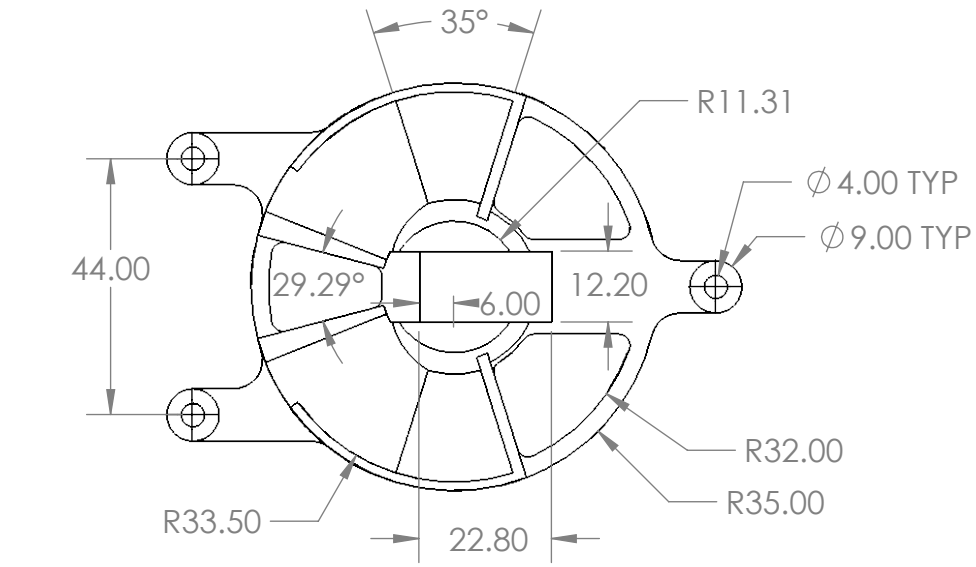
UNLESS OTHERWISE SPECIFIED:		NAME	DATE
DIMENSIONS ARE IN MILLIMETERS	DRAWN		
	CHECKED		
	ENG APPR.		
	MFG APPR.		
INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
MATERIAL PLA PLASTIC	COMMENTS: 3D PRINTED, NO CLEANUP NECESSARY		
FINISH PRINT FINISH			
DO NOT SCALE DRAWING			

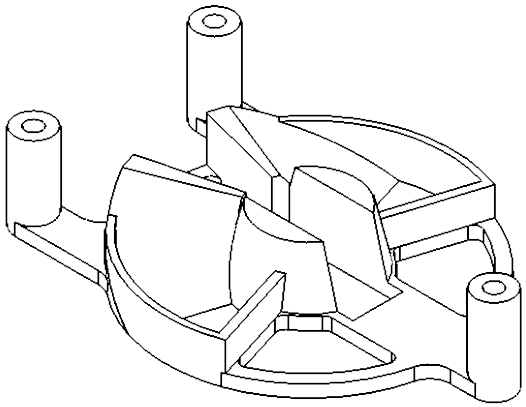
TITLE:

PART 6

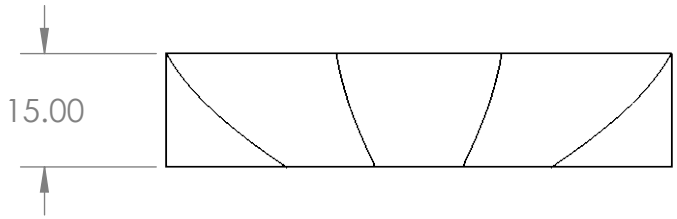
SIZE	DWG. NO.	REV
A		

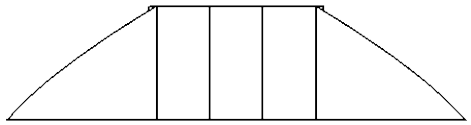
SCALE: 1:1

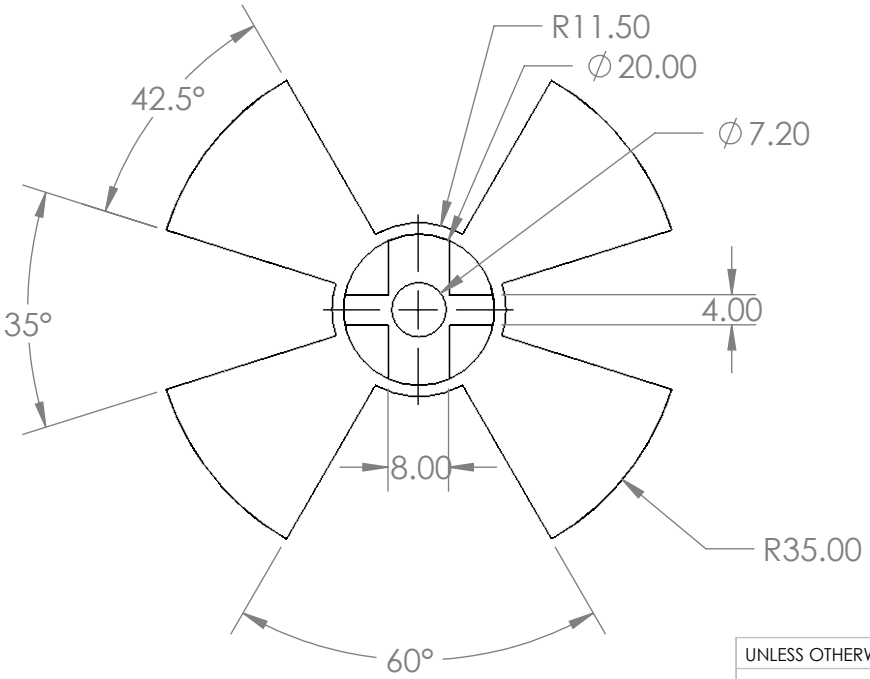




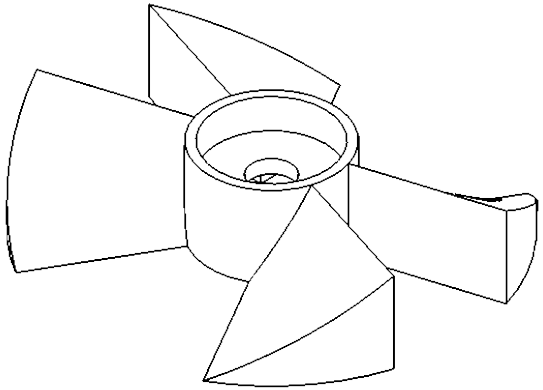
UNLESS OTHERWISE SPECIFIED:		NAME	DATE	
DIMENSIONS ARE IN MILLIMETERS	DRAWN			
	CHECKED			TITLE: PART 7
	ENG APPR.			
	MFG APPR.			
INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.			
MATERIAL PLA PLASTIC	COMMENTS: 3D PRINTED, NO CLEANUP NECESSARY			SIZE A / DWG. NO. / REV
FINISH PRINT FINISH				
DO NOT SCALE DRAWING				SCALE: 1:2



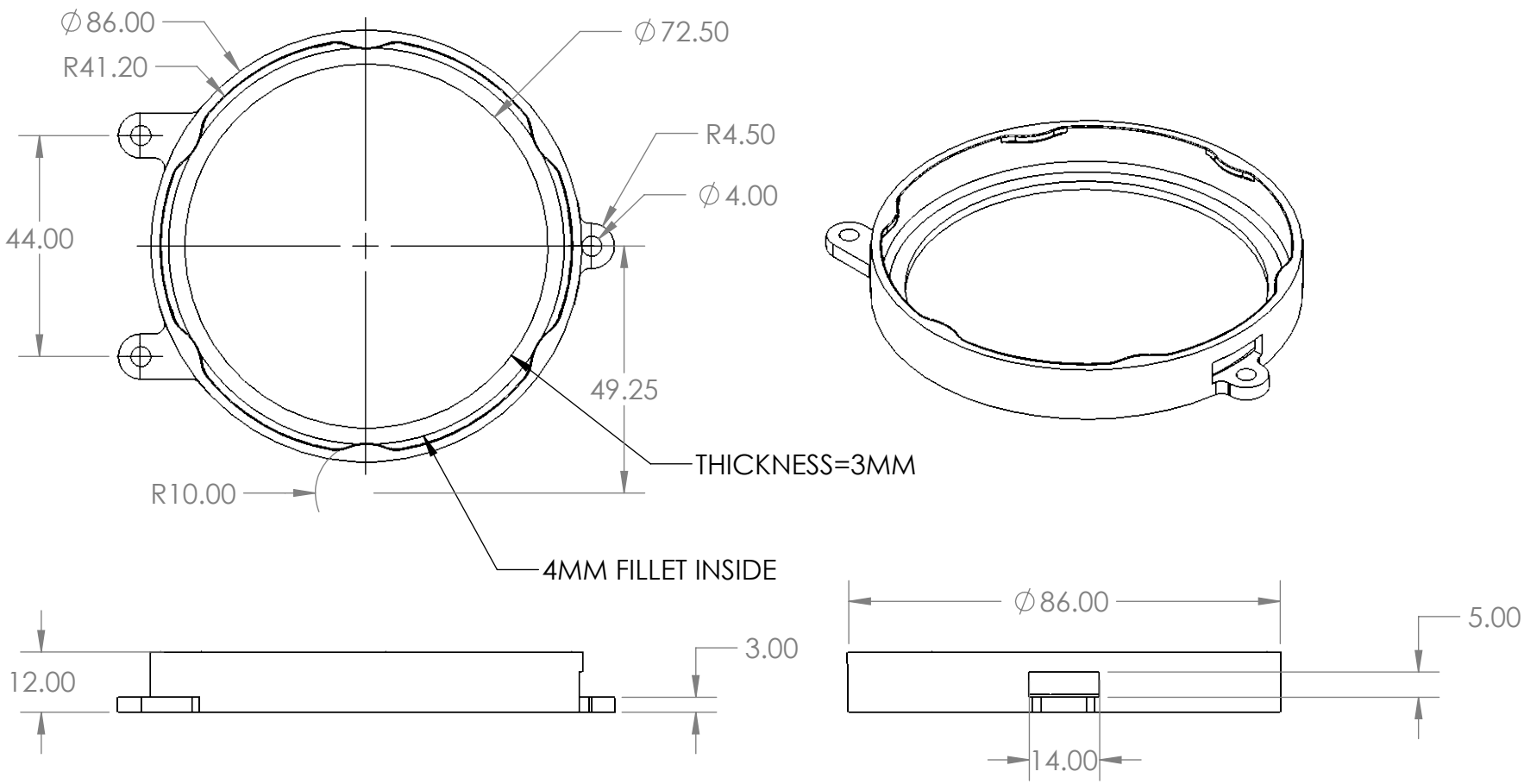




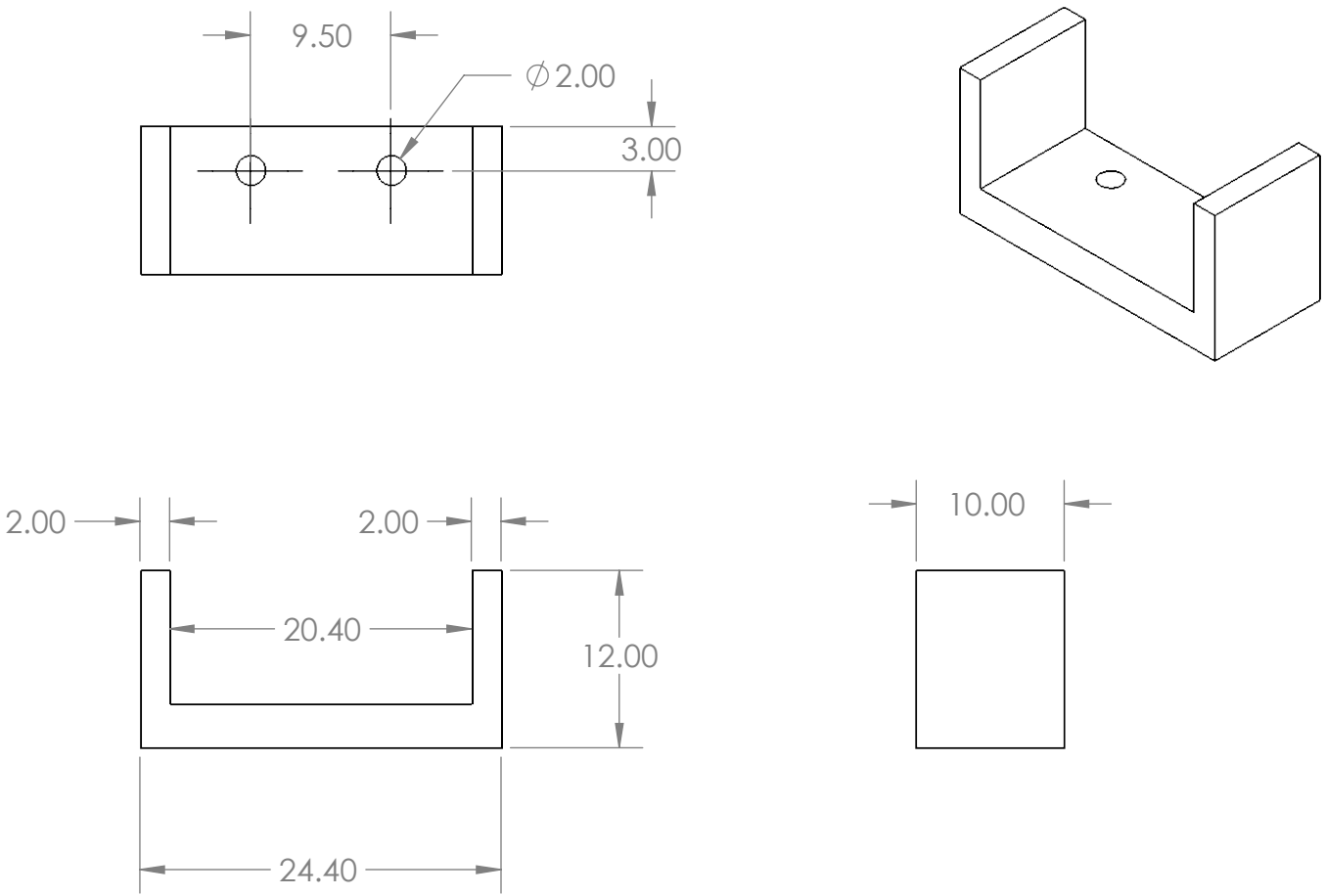
BOTTOM VIEW



UNLESS OTHERWISE SPECIFIED:		NAME	DATE	
DIMENSIONS ARE IN MILLIMETERS	DRAWN			
	CHECKED			TITLE: PART 8
	ENG APPR.			
	MFG APPR.			
INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.			
MATERIAL PLA PLASTIC	COMMENTS: 3D PRINTED, NO CLEANUP NECESSARY			SIZE A / DWG. NO. / REV
FINISH PRINT FINISH				
DO NOT SCALE DRAWING				SCALE: 1:1



UNLESS OTHERWISE SPECIFIED:		NAME	DATE	
DIMENSIONS ARE IN MILLIMETERS	DRAWN			
	CHECKED			TITLE: PART 9
	ENG APPR.			
	MFG APPR.			
INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.			
MATERIAL PLA PLASTIC	COMMENTS: 3D PRINTED, NO CLEANUP NECESSARY			SIZE A / DWG. NO. / REV
FINISH PRINT FINISH				
DO NOT SCALE DRAWING				SCALE: 1:2

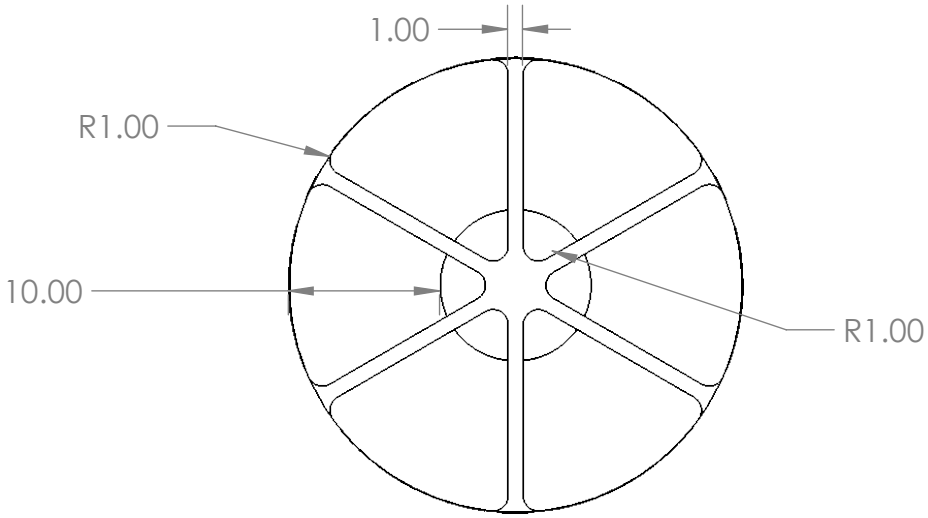


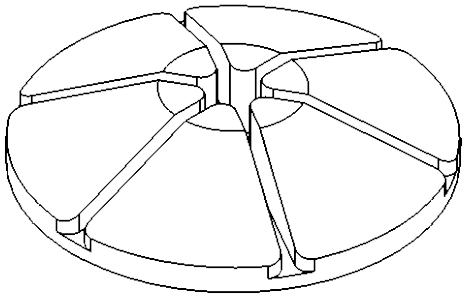
UNLESS OTHERWISE SPECIFIED:		NAME	DATE
DIMENSIONS ARE IN MILLIMETERS	DRAWN		
	CHECKED		
	ENG APPR.		
	MFG APPR.		
INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
MATERIAL PLA PLASTIC	COMMENTS: 3D PRINTED, NO CLEANUP NECESSARY		
FINISH PRINT FINISH			
DO NOT SCALE DRAWING			

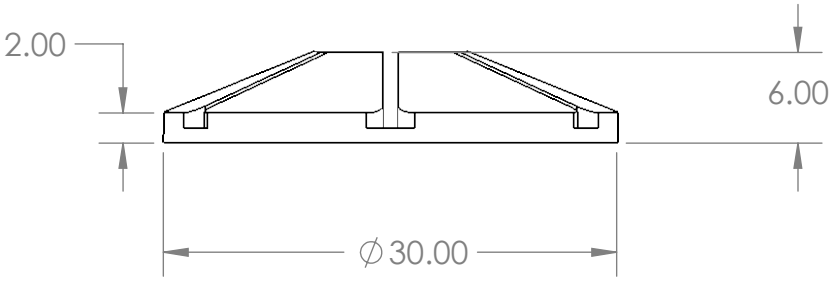
TITLE: PART 10

SIZE	DWG. NO.	REV
A		

SCALE: 2:1







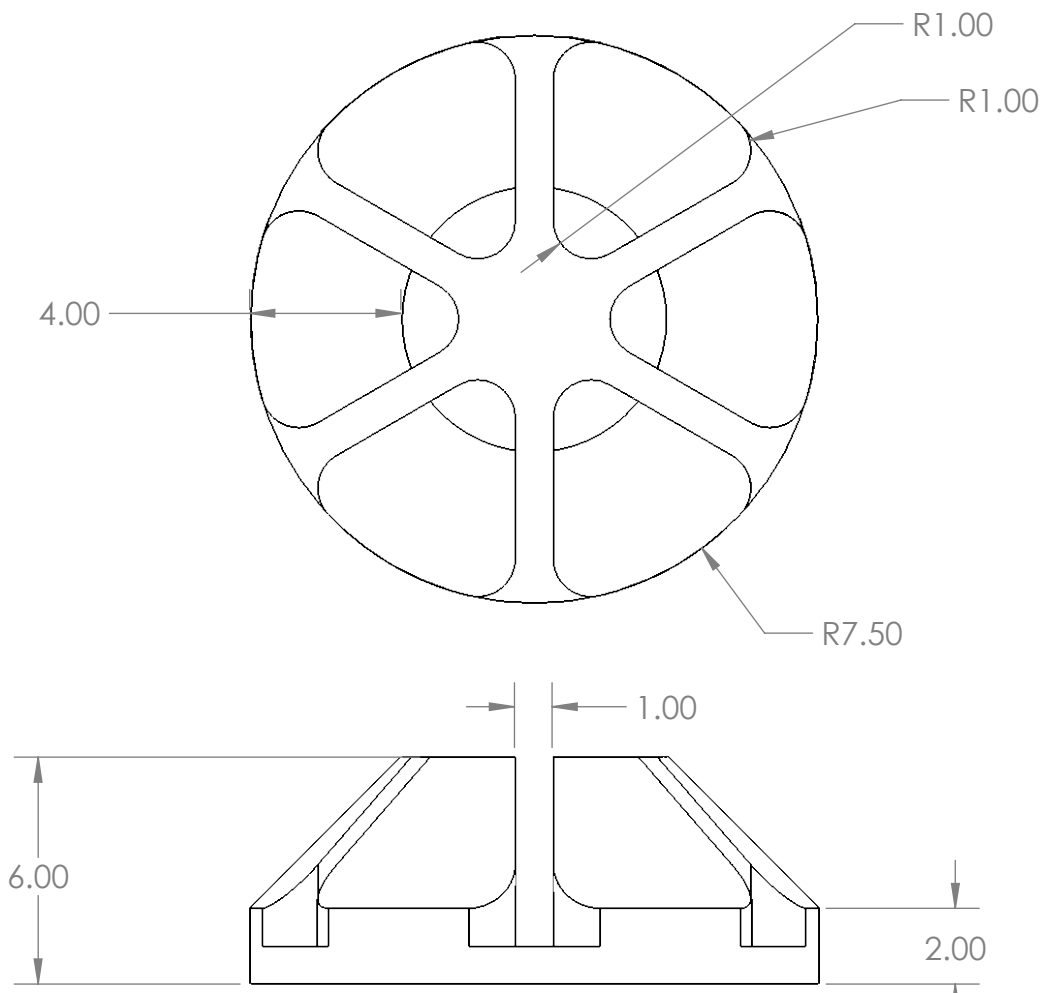
UNLESS OTHERWISE SPECIFIED:		NAME	DATE
DIMENSIONS ARE IN MILLIMETERS	DRAWN		
	CHECKED		
	ENG APPR.		
	MFG APPR.		
INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
MATERIAL PLA PLASTIC	COMMENTS: 3D PRINTED, NO CLEANUP NECESSARY		
FINISH PRINT FINISH			
DO NOT SCALE DRAWING			

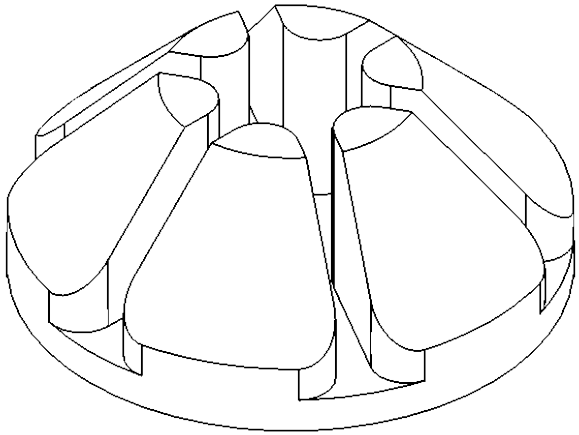
TITLE:

PART 11

SIZE	DWG. NO.	REV
A		

SCALE: 2:1



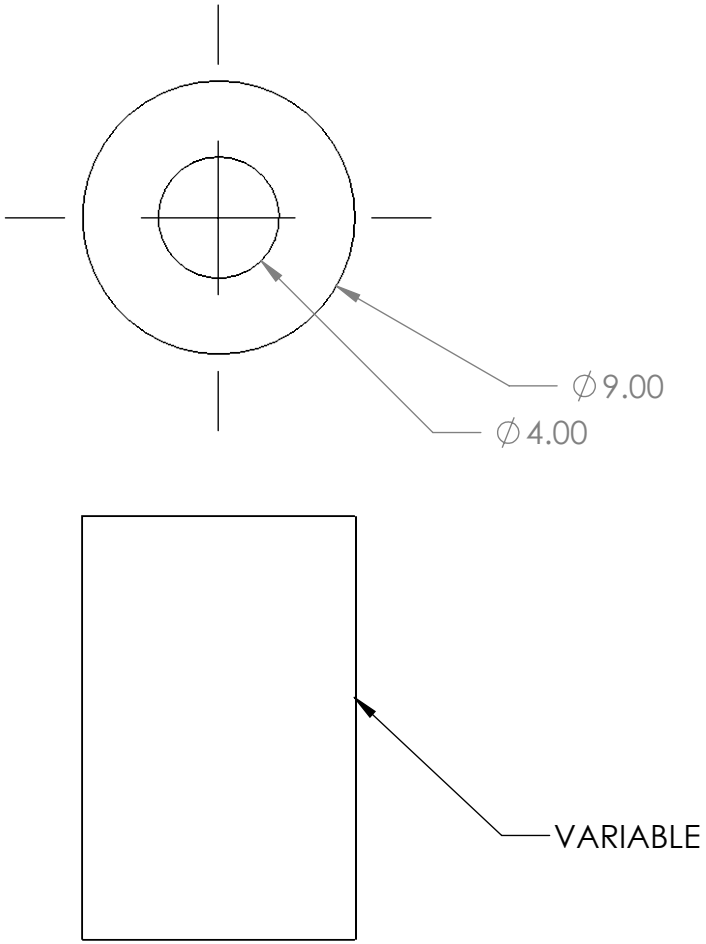


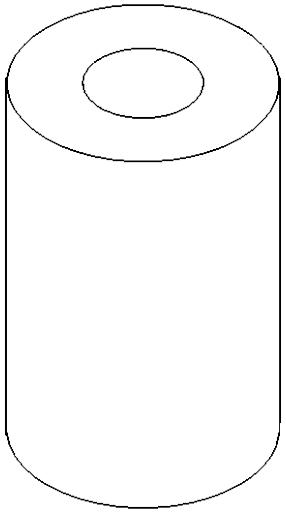
UNLESS OTHERWISE SPECIFIED:		NAME	DATE
DIMENSIONS ARE IN MILLIMETERS	DRAWN		
	CHECKED		
	ENG APPR.		
	MFG APPR.		
INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
MATERIAL PLA PLASTIC	COMMENTS: 3D PRINTED, NO CLEANUP NECESSARY		
FINISH PRINT FINISH			
DO NOT SCALE DRAWING			

TITLE: PART 12

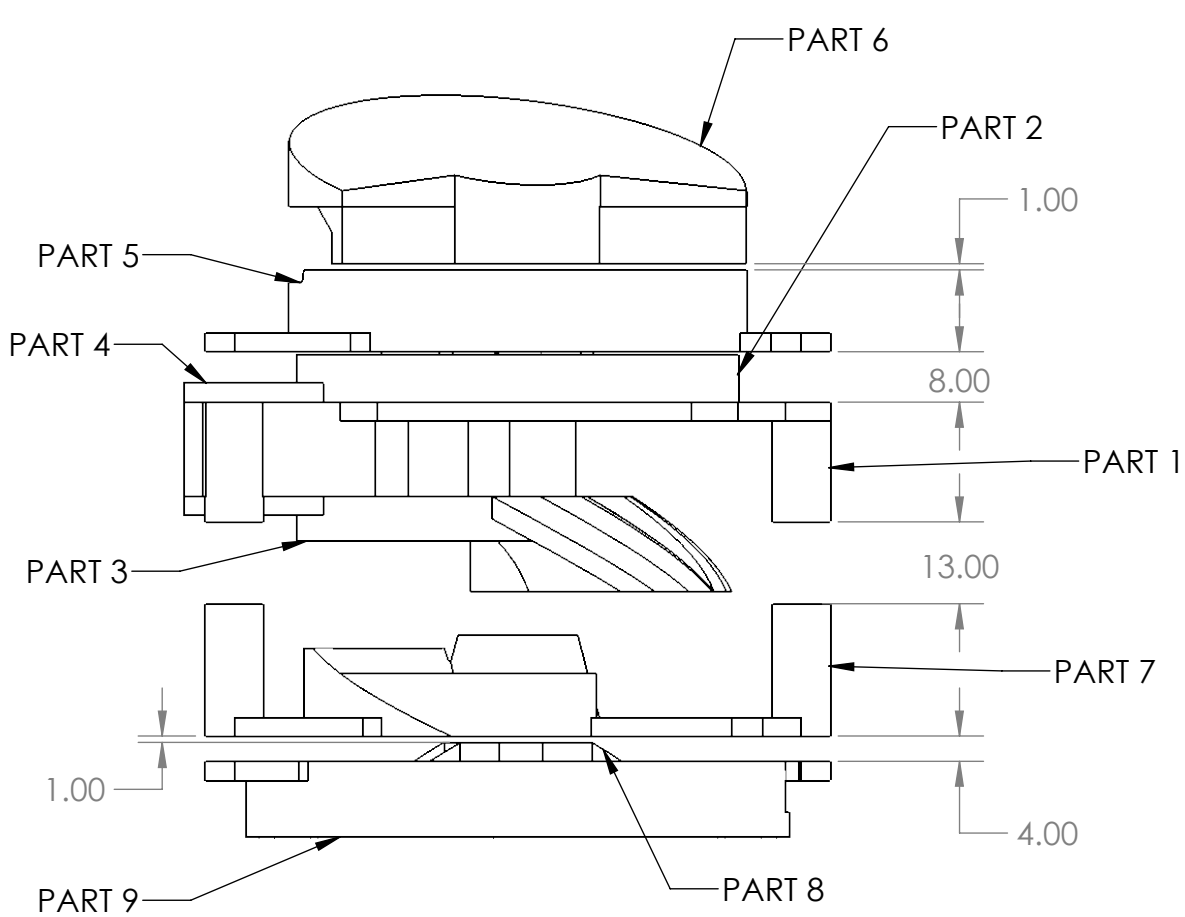
SIZE	DWG. NO.	REV
A		

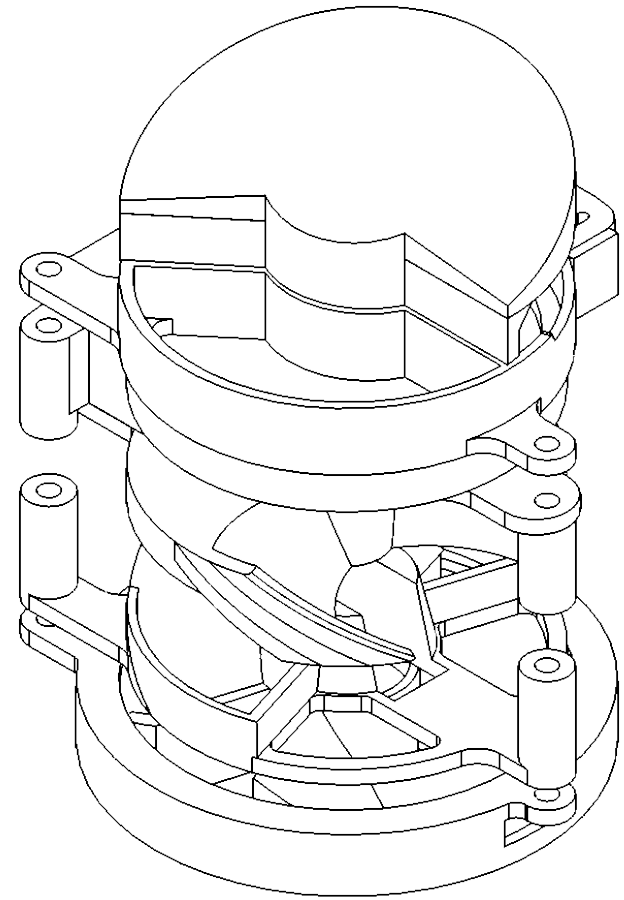
SCALE: 5:1





UNLESS OTHERWISE SPECIFIED:		NAME	DATE	
DIMENSIONS ARE IN MILLIMETERS	DRAWN			
	CHECKED			TITLE: SPACER EXAMPLE
	ENG APPR.			
	MFG APPR.			
INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.			
MATERIAL PLA PLASTIC	COMMENTS: 3D PRINTED, NO CLEANUP NECESSARY			SIZE A / DWG. NO. / REV
FINISH PRINT FINISH				
DO NOT SCALE DRAWING				SCALE: 5:1





UNLESS OTHERWISE SPECIFIED:		NAME	DATE
DIMENSIONS ARE IN MILLIMETERS	DRAWN		
	CHECKED		
	ENG APPR.		
	MFG APPR.		
INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
MATERIAL PLA PLASTIC	COMMENTS: 3D PRINTED, NO CLEANUP NECESSARY		
FINISH PRINT FINISH			
DO NOT SCALE DRAWING			

TITLE:

ASSEMBLY OF PARTS 1-10

SIZE	DWG. NO.	REV
A		

SCALE: 1:2