Volcano Bot

By: Neel Patel and Darsh Patel

Questions

Can we create a prototype of a robot that can travel around a volcano to collect rock samples, and data so scientists do not have to risk their lives doing so?

Hypothesis

Our hypothesis is that this robot will be able to move, collect samples, and data about the gases in the air.

Purpose

We are trying to create a robot that can collect samples and data which can be used for research purposes by scientists.

Background Information

Volcano Gases

Volcanoes are best known for huge explosions of lava and rock, but most dormant and active volcanoes will emit a constant stream of gases that can be just as informative as the lava itself:

Carbon Dioxide:

Carbon Dioxide (C02) is released from volcanoes in two ways: eruptions, and more commonly through vents, porous rock, and soil which leak built up carbon dioxide from underground magma chambers. Increased levels of C02 at the surface can indicate magma from deep below are entering the volcanic system.

Sulfur Dioxide:

Sulfur Dioxide (S02) is emitted from a volcano when magma is near the surface. High levels of Sulfur Dioxide can indicate an eruption although sulfur dioxide easily dissolves in water and if any lakes or bodies of water are near the volcano it can be impossible to detect the actual amounts of Sulfur Dioxide being released.

Hydrogen Sulfide:

Hydrogen Sulfide (H2S) is created when sulfur gases from the volcano reacts with water from groundwater. The presence of hydrogen sulfide can indicate that the volcano is relatively quiet as it means that the groundwater is filtering out the sulfur gas.

Carbon Monoxide:

Carbon Monoxide (CO) is another gas released from a volcano. High levels of carbon Monoxide indicate an eruption.

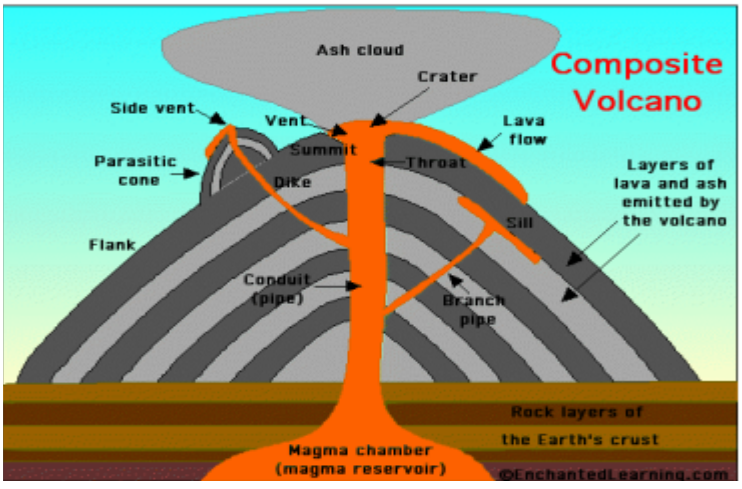
Temperature:

High temperatures around a volcano can also mean that the volcano is active.

Volcano monitoring is important because volcanic eruptions can cause serious damage to the health of humans, such as the sulfur which is released in the atmosphere can cause breathing problems in both healthy and people with respiratory issues, and it is important to be able to monitor when volcanoes will erupt so the proper precautions can be taken, like getting people to evacuate the area before the eruption. It is important to collect lava samples as it can provide information on how hot the magma was before the volcano erupted through how much magnesium is present in it. With the data collected scientists will also be able to determine the effects, such as the plumes of sulfur it released in the atmosphere with can block sun rays from reaching the Earth which will have a devastating effect on plants and animals.

Active volcanoes are volcanoes that have exploded in the last 10 000 years while a dormant volcano is an active volcano that is supposed to erupt but has not erupted yet, and an extinct volcano is a volcano that has not erupted in the last 10 000 years.

Volcanoes provide information on Earth's interior. If we made advancements to our model our real robot would mostly travel around the base, flanks, although if need be it could also travel near the summit if needed. (diagram below) The flank of the volcano is where the lava flow would be,and also where most of the rock samples would be.



There are three main types of volcanoes, composite, shield, and dome. Composite volcanoes are steep cones formed by ash and lava flows. Composite volcanoes, such as Mount St. Helens, are also likely to have pyroclastic flows rather than lava flows. Pyroclastic flows are a mixture of super hot steam, ash, and rock. Shield volcanoes, such as Kīlauea, are volcanoes formed by layers of lava, they have gently sloping sides and are relatively low to the ground. They are not explosive but rather produce lava that can flow for many miles,

though it rarely results in death or injury. Dome volcanoes, Mount Merapi, are shaped like a dome as a result of their lava being really viscous, this results in the lava cooling before it makes it to the ground. Out of the three main types of volcanoes our volcano would be best suited for a shield volcano as the gently sloping sides would let our robot move easily throughout the terrain.

Why is it bad for Scientists:

Avalanches:

Much like mountains volcanoes also have avalanches. Rapid growth of the cone can lead to steep sided and unstable volcanoes. Rising magma, earthquakes and heavy rain can trigger debris avalanches at anytime. This creates a dangerous setting for any scientist trying to collect samples from the volcano. These avalanches can either be wet or dry, but if they are wet they can evolve and continue to flow down slope even farther as a mudslide.

Carbon Dioxide:

On average, volcanoes release about 180 and 440 million tonnes of carbon dioxide into the atmosphere. It currently constitutes about 0.04% of the air in the world, but as more cold and heavier carbon dioxide falls to the ground, it can build up concentration in low lying areas such as crevasses and gaps in the slopes of a volcano. A concentration of 3% can lead to dizziness and breathing trouble, but as it quickly increases to about 15% it can cause unconsciousness and soon after, death. Volcanic eruptions can expel upwards of 480,000 ppm of carbon dioxide with the average being 160,000 ppm, both concentrations high enough to cause death.

Sulfur Dioxide:

Sulfur dioxide is a colourless yet pungent gas that is expelled from volcanoes. It is very irritable and burns your eyes, skin and the inner lining of your throat. Inhalation of this gas is very toxic and at 100 ppm it is considered life-threatening. On average volcanic gas spewed from a volcano consist of 177,000 ppm of sulfur dioxide but this concentration can reach up to 477,000 ppm.

Carbon Monoxide:

Carbon Monoxide is an odorless, colourless, and tasteless gas. Breathing in high levels of this substance can kill while lower doses can permanently damage your heart and brain. Symptoms of carbon monoxide poisoning include headache, dizziness, nausea, fatigue, weakness and shortness of breath.

Hydrogen Sulfide:

As a colorless and flammable gas hydrogen sulfide has a very strong rotten egg smell. The gas is very toxic and can cause irritation of the respiratory system, and during long exposures it can cause fluid to build up the lungs. Exposure over 500 ppm can cause unconsciousness in less than 5 minutes and can cause death in under an hour. Volcanic

eruptions can expel upwards of 11,200 ppm of Hydrogen Sulfide with the average being 6,100 ppm, both concentrations high enough to cause death.

Crevasses:

Crevasses and gaps in the terrain of the volcano could be extremely dangerous towards scientists, especially if the volcano was near the poles where snow and precipitation would cover the terrain.

Time of Flight sensors (ToF)

ToF sensors or (time of flight sensors) capture a 3D image of its surroundings. They work by first illuminating the scene with pulsing or continuous light than observing the time it takes for the light to reach the object than back again. Since to speed of light is relatively constant, you can calculate the distance to certain points on the scene. Combining this data with a regular camera you can create a textured 3D graphic.

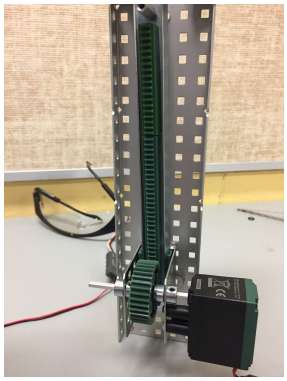
Materials

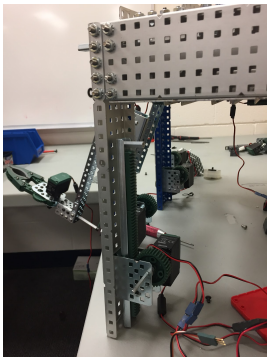
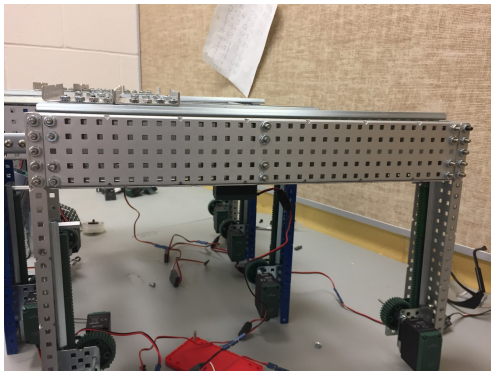
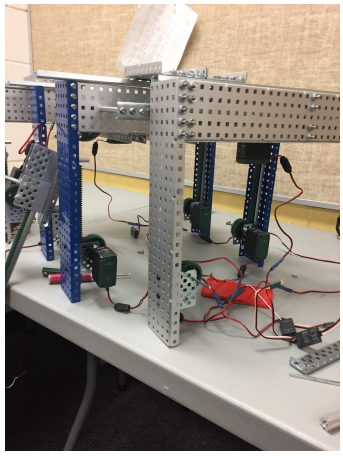
- Vex Robotics Components
- Raspberry Pi 3B+ with Wifi
- Arduino Uno “Brain”
- BreadBoard
- Temperature sensor
- Vex Robotics Brain
- Vex robotics Remote
- Mq-135 sensor
- Mq-136 sensor
- Mq-7 sensor
- Ultrasonic Sensor
- DHT11 Humidity and Temperature Sensor
- Kuman Camera Module

Procedure

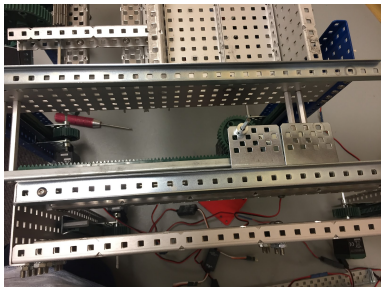
Variables

- Manipulate Variable: The robot itself e.g. Having pegs instead of wheels
- Responding Variable: How well suited the robot will be for the environment
- Controlled Variables: The environment the robot is in, for our purposes, it is inside a volcano

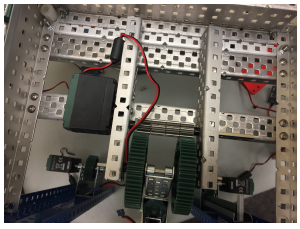
1. Create a Linear Motion Mechanism by first placing an Outer Acetal Slide Truck into a Rack Gearbox Bracket then use small motor screws to screw a 393 2-wire motor onto one of the top level holes in its side. Then place a shaft into the motor with a 36 tooth gear in it, but make sure to place shaft collars on all ends, and spacers between the bracket and the gear so. Then create use small motor screws to screw rack gears all the way across onto a 7.5” slide track. Then insert this track into the Outer Acetal Slide Truck.
2. Then use 0.25” screws to screw this Linear Motion Mechanism onto a 12.5” by 2.5” C-Channel. This will become one of our legs for the robot. 
3. Create 8 of what was described in Step 2
4. Get a 17.5” by 2.5” C-Channel and screw a 17.5” by 1” C-Channel onto its two sides using 1.5” screw and keps nuts into 5 places- two on both sides and one in the middle. These will become our big side connectors
5. Create 4 copies of what was said in step 4
6. Screw two legs onto to both sides of one side connectors using 0.5” screws and keps nuts. Screw them into all 5 available holes.

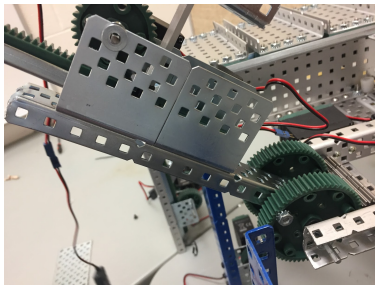
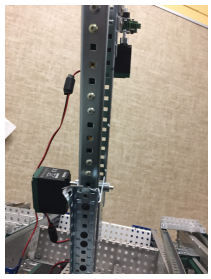
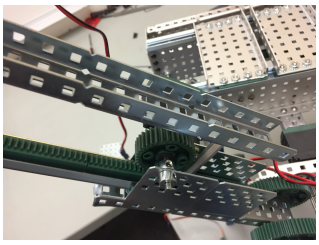
7. Create another copy of what was said in step 6
8. Create another Linear Motion Mechanism but use instead this time a 12” rail but keep two spaces on either end where you will insert 2 4” standoffs on either end.
9. Screw the Linear Motion Mechanism onto the middle of the big side connector using a 0.5” screw.



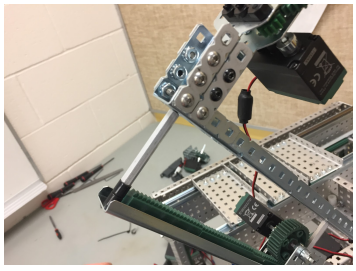
10. Get a 15” by 2.5” C-Channel and screw two more legs onto its sides. This is what we call a small side connector.
11. Create another copy of what was described in step 9
12. Connect the two small side connectors with two 10” by 2.5” C-Channels side by side, using 0.375 screws and keps nuts.
13. Then on the sides of the C-Channels described in step 12 attach two 9” by 2.5” C-Channels vertically in the middle to create a bin for our samples.
14. Pair two 12” by 1” C-Channels side by side and on the last hole put in a shaft with two 60 tooth gears in it. 0.5” behind the two 60 tooth gears place a shaft collar and on the shaft collar place 4 12-tooth metal gears. Screw in a motor on the shaft collar with the 4 12-tooth metal gears using motor screws. This will be known as the base of our claw.



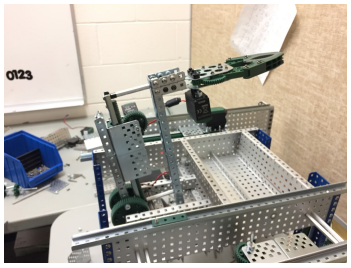
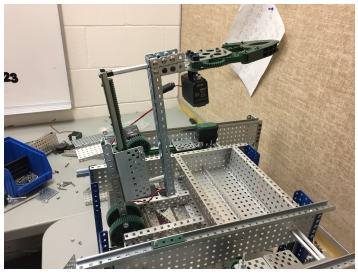
15. Between the middle of the 60-tooth gears screw in a 6.5” by 1” C-Channel. On that C-Channel place a Linear Motion Mechanism but for the rail use a 8.5” rail but when screwing in the rack gears leave two spaces on either end. For those spaces put in 2” standoffs but when screwing into the standoffs use a small spacer.

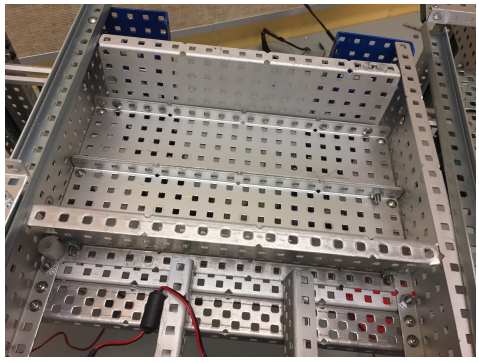
16. Get a premade claw and attach it to 10” by 1” C-Channel using 2 1x4 bars.



17. Attach that C-Channel to the standoffs in the Rail described in step 15. This whole thing is what we like the call the full claw
18. Attach a 10” by 1” C-Channel between the two small connectors beside the two big C-Channels already present. On that small C-Channel attach the full claw on to it using screws.

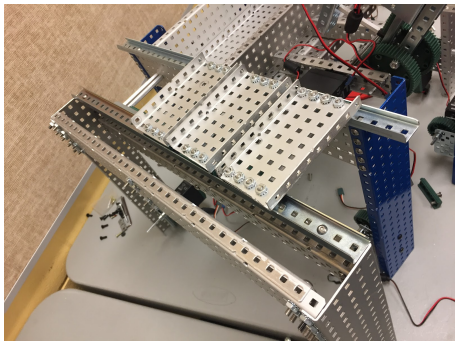



19. Attach another 10” by 1” C-Channel between the small side connectors right under the small 12 tooth gears described in step 14. And attach the full claw to this C-Channel. Make sure that there is enough space between the two C-Channels described in step 19 and 18 for the claw to fully turn around, and be able to drop an item into bin for the samples.

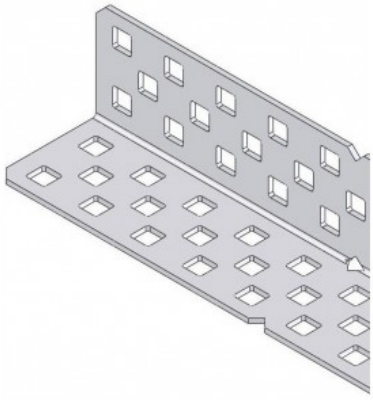


20. Now you can connect the small side connector and the big side connector by screwing in the standoffs described in step 8 into the small side connector.
21. On top of the big side, connectors attach a 17.5” rail using a screw on the second last screw hole on one side. Do the same with the smaller connectors but for the excess keep three of the extra holes in the front and 2 in the back.

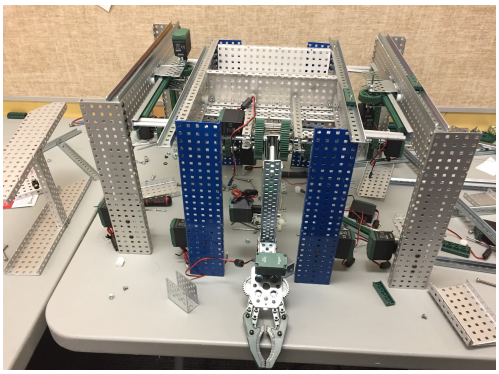
22. Using two Inner Acetal Slide Truck attach a 4.5” by 2.5” C-Channel between them using 0.5” screws and keps nuts.
23. Make 6 copies of what was described in step 22
24. Put three of the objects described in step 22 in the rails between one small and one big side connector, and do the same with the other pair.



25. Now, remember the two copies of the object described in step 4 that are left. We will now attach those on to legs of the big side connectors like we did in step 6.
26. Now attach an 18x5 metal plate to a 17.5” by 2.5” C-Channel by using L-Brackets to attach the two at a 90 degrees perpendicular angle.
27. Do the same to the other side of the metal plate by attaching it to another 17.5” by 2.5” C-Channel by using L-Brackets to attach the two at a 90 degrees perpendicular angle.
28.



29. Now attach this whole placer onto the C-Channel described in step 19.
30. Now rest the camera module onto this metal plate.
31. Now we will attach our sensors, raspberry pi’s, and breadboard, and our VEX brains to this metal plate. The VEX brain will be placed upside down underneath the collection bin. The sensors and raspberry pi circuit will also go where the camera was mounted.



1st Controller

*Everything colour coded the same has the same function though the values or names may be different

```
#pragma config(Motor,  port1,           lateralmotor1, tmotorVex393_HBridge, openLoop)
#pragma config(Motor,  port2,           leftpegs1,     tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port3,           leftpegs2,     tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port4,           leftpegs3,     tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port5,           centralpegs1,  tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port6,           centralpegs2,  tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port7,           centralpegs3,  tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port8,           centralpegs4,  tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port10,          lateralmotor2, tmotorVex393_HBridge, openLoop)
```

- This part of the code tells the brain which motors plug into which ports

```
task main()
```

- Tells the brain that the code inside the yellow brackets

```
{
  while(1 == 1)
```

- Tells the brain that the code inside the orange brackets is to be followed only when 1 = 1

```
  {
    motor[lateralmotor1] = vexRT[Ch2] / 2;
    motor[lateralmotor2] = vexRT[Ch3] / 2;
```

- This part of the code says that the motors given the name latermotor1 and lateral motor 2 are to be moved by channel 2&3 on the controller

```
    if(vexRT[Btn5U] == 1)
```

- This part of the code says that when button 5U is pressed (or when it “equals one”) to move the motors in the respective directions and with the respective power

```
    {
      motor[leftpegs1] = 127;
      motor[leftpegs2] = -127;
      motor[leftpegs3] = 127;
    }
    else if(vexRT[Btn5D] == 1)
```

- # = power and - means counter-clockwise and + or putting nothing in front of the number means it will move clockwise
- This part of the code says when button 5D is pressed to move the motors in the respective directions and with the respective power

```
    {
      motor[leftpegs1] = -127;
      motor[leftpegs2] = 127;
      motor[leftpegs3] = -127;
    }
    else
    {
      motor[leftpegs1] = 0;
      motor[leftpegs2] = 0;
      motor[leftpegs3] = 0;
    }
```

- This part tells you that when nothing is pressed the motore will be given 0 power meaning that the motors do not move

```
    if(vexRT[Btn6U] == 1)
```

- This part of the code says that when button 6U is pressed (or when it “equals one”) to move the motors in the respective directions and with the respective power

```
    {
      motor[centralpegs1] = 127;
      motor[centralpegs2] = -127;
      motor[centralpegs3] = 127;
      motor[centralpegs4] = -127;
    }
    else if(vexRT[Btn6D] == 1)
```

- This part of the code says when button 6D is pressed to move the motors in the respective directions and with the respective power

```
    {
      motor[centralpegs1] = -127;
      motor[centralpegs2] = 127;
      motor[centralpegs3] = -127;
      motor[centralpegs4] = 127;
    }
    else
    {
      motor[centralpegs1] = 0;
      motor[centralpegs2] = 0;
      motor[centralpegs3] = 0;
      motor[centralpegs4] = 0;
    }
  }
}
```

- This part tells you that when nothing is pressed the motore will be given 0 power meaning that the motors do not move

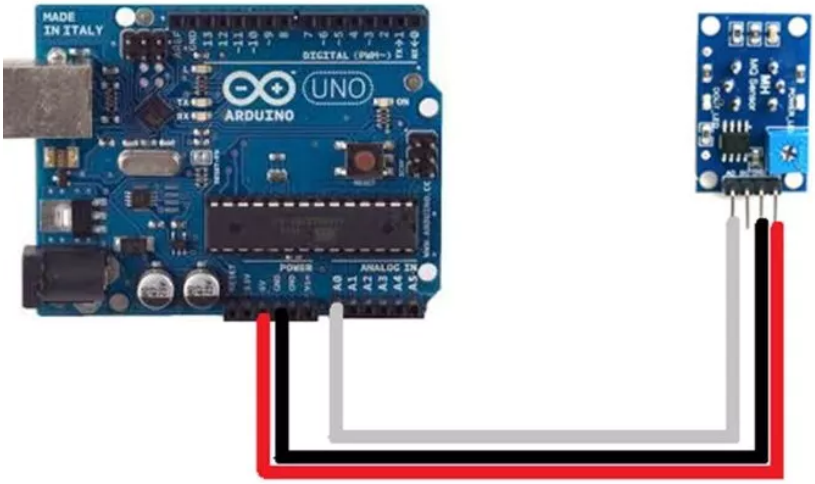
2nd Controller

```
#pragma config(Motor,  port1,           lateralmotor1, tmotorVex393_HBridge, openLoop)
#pragma config(Motor,  port2,           rightpegs4,    tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port4,           rightpegs1,    tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port5,           rightpegs2,    tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port6,           rightpegs3,    tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port7,           claw,          tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port8,           arm,           tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port10,          lateralmotor2, tmotorVex393_HBridge, openLoop)

task main()
{
  while(1 == 1)
  {
    motor[lateralmotor1] = vexRT[Ch3] / 2;
    motor[lateralmotor2] = vexRT[Ch2] / 2;

   if(vexRT[Btn5U] == 1)
   {
     motor[rightpegs1] = -127;
     motor[rightpegs2] = 127;
     motor[rightpegs3] = 127;
     motor[rightpegs4] = -127;
    }
    else if(vexRT[Btn5D] ==1)
    {
      motor[rightpegs1] = 127;
      motor[rightpegs2] = -127;
      motor[rightpegs3] = -127;
      motor[rightpegs4] = 127;
    }
    else
    {
      motor[rightpegs1] = 0;
      motor[rightpegs2] = 0;
      motor[rightpegs3] = 0;
      motor[rightpegs4] = 0;
    }

     if(vexRT[Btn6U] == 1)
     {
       motor[arm] = 127;
     }
     else if(vexRT[Btn6D] == 1)
     {
       motor[arm] = -127;
      }
      else
      {
      motor[arm] = 0;
      }

       if(vexRT[Btn8D] == 1)
       {
         motor[claw] = -127;
       }
         else
       {
         motor[claw] = 0;
        }
        if(vexRT[Btn8U] == 1)
        {
```

```
            motor[claw] = 127;
          }
          else
          {
            motor[claw] = 0;
          }
        }
      }
```

:

MQ-135



Connect the Arduino to the raspberry the Arduino USB Cable.
Install Arduino IDE on to your raspberry PI using this command on the SSH terminal
sudo apt-get update && sudo apt-get upgrade
sudo apt-get install arduino
Open Arduino IDE on your raspberry Pi and type:

```
#define         MQ_PIN                      (0)     //define which analog input channel you are going to use
#define         RL_VALUE                    (5)     //define the load resistance on the board, in kilo ohms
#define         RO_CLEAN_AIR_FACTOR         (9.83)  //RO_CLEAR_AIR_FACTOR=(Sensor resistance in clean air)/RO,
                                            //which is derived from the chart in datasheet

/***********************Software Related Macros************************************/
#define         CALIBARAION_SAMPLE_TIMES    (50)    //define how many samples you are going to take in the calibration
phase
#define         CALIBRATION_SAMPLE_INTERVAL (500)   //define the time interal(in milisecond) between each samples in the
                                            //cablibration phase
#define         READ_SAMPLE_INTERVAL        (50)    //define how many samples you are going to take in normal operation
#define         READ_SAMPLE_TIMES           (5)     //define the time interal(in milisecond) between each samples in
#include <LiquidCrystal.h>

const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);                                      //normal operation

/*********************Application Related Macros*********************************/
#define         GAS_LPG                     (0)
#define         GAS_CO                      (1)
#define         GAS_SMOKE                   (2)

/****************************Globals**********************************************/
float           LPGCurve[3]  =  {2.3,0.21,-0.47};   //two points are taken from the curve.
                                            //with these two points, a line is formed which is "approximately equivalent"
                                            //to the original curve.
                                            //data format:{ x, y, slope}; point1: (lg200, 0.21), point2: (lg10000, -0.59)
float           COCurve[3]  =  {2.3,0.72,-0.34};    //two points are taken from the curve.
                                            //with these two points, a line is formed which is "approximately equivalent"
                                            //to the original curve.
                                            //data format:{ x, y, slope}; point1: (lg200, 0.72), point2: (lg10000,  0.15)
float           SmokeCurve[3] ={2.3,0.53,-0.44};    //two points are taken from the curve.
                                            //with these two points, a line is formed which is "approximately equivalent"
                                            //to the original curve.
                                            //data format:{ x, y, slope}; point1: (lg200, 0.53), point2: (lg10000,  -0.22)
float           Ro           =  10;                 //Ro is initialized to 10 kilo ohms

void setup()
{
  Serial.begin(9600);                           //UART setup, baudrate = 9600bps
  Serial.print("Calibrating...\n");
  Ro = MQCalibration(MQ_PIN);                       //Calibrating the sensor. Please make sure the sensor is in clean air
   lcd.begin(16, 2);                                    //when you perform the calibration
  Serial.print("Calibration is done...\n");
  Serial.print("Ro=");
  Serial.print(Ro);
  Serial.print("kohm");
  Serial.print("\n");
lcd.print("Calibration is done...\n");
lcd.print("Ro=");
lcd.print(Ro);
lcd.print("kohm");
lcd.print("\n");
}

void loop()
{
   Serial.print("LPG:");
   Serial.print(MQGetGasPercentage(MQRead(MQ_PIN)/Ro,GAS_LPG) );
   Serial.print( "ppm" );
   Serial.print("    ");
   Serial.print("CO:");
   Serial.print(MQGetGasPercentage(MQRead(MQ_PIN)/Ro,GAS_CO) );
   Serial.print( "ppm" );
   Serial.print("    ");
   Serial.print("SMOKE:");
   Serial.print(MQGetGasPercentage(MQRead(MQ_PIN)/Ro,GAS_SMOKE) );
   Serial.print( "ppm" );
   Serial.print("\n");
       lcd.setCursor(0, 0);
       lcd.print("LPG:");
```

```
lcd.print(MQGetGasPercentage(MQRead(MQ_PIN)/Ro,GAS_LPG) );
//lcd.print( "ppm" );
lcd.print("     ");
  lcd.setCursor(9, 0);
lcd.print("CO:");
lcd.print(MQGetGasPercentage(MQRead(MQ_PIN)/Ro,GAS_CO) );
//lcd.print( "ppm" );
lcd.print("       ");
 lcd.setCursor(0, 1);
lcd.print("SMOKE:");
lcd.print(MQGetGasPercentage(MQRead(MQ_PIN)/Ro,GAS_SMOKE) );
//lcd.print( "ppm" );
lcd.print("         ");
   delay(200);
}

/**************** MQResistanceCalculation ************************************
Input:   raw_adc - raw value read from adc, which represents the voltage
Output:  the calculated sensor resistance
Remarks: The sensor and the load resistor forms a voltage divider. Given the voltage
         across the load resistor and its resistance, the resistance of the sensor
         could be derived.
****************************************************************************/
float MQResistanceCalculation(int raw_adc)
{
  return ( ((float)RL_VALUE*(1023-raw_adc)/raw_adc));
}

/***************************** MQCalibration ************************************
Input:   mq_pin - analog channel
Output:  Ro of the sensor
Remarks: This function assumes that the sensor is in clean air. It use
         MQResistanceCalculation to calculates the sensor resistance in clean air
         and then divides it with RO_CLEAN_AIR_FACTOR. RO_CLEAN_AIR_FACTOR is about
         10, which differs slightly between different sensors.
****************************************************************************/
float MQCalibration(int mq_pin)
{
  int i;
  float val=0;

  for (i=0;i<CALIBARAION_SAMPLE_TIMES;i++) {            //take multiple samples
    val += MQResistanceCalculation(analogRead(mq_pin));
    delay(CALIBRATION_SAMPLE_INTERVAL);
  }
  val = val/CALIBARAION_SAMPLE_TIMES;                  //calculate the average value

  val = val/RO_CLEAN_AIR_FACTOR;                       //divided by RO_CLEAN_AIR_FACTOR yields the Ro
                                        //according to the chart in the datasheet

  return val;
}
/*****************************  MQRead *********************************************
Input:   mq_pin - analog channel
Output:  Rs of the sensor
Remarks: This function use MQResistanceCalculation to caculate the sensor resistenc (Rs).
         The Rs changes as the sensor is in the different consentration of the target
         gas. The sample times and the time interval between samples could be configured
         by changing the definition of the macros.
****************************************************************************/
float MQRead(int mq_pin)
{
  int i;
  float rs=0;

  for (i=0;i<READ_SAMPLE_TIMES;i++) {
    rs += MQResistanceCalculation(analogRead(mq_pin));
    delay(READ_SAMPLE_INTERVAL);
  }

  rs = rs/READ_SAMPLE_TIMES;
```

```
  return rs;
}

/***************************  MQGetGasPercentage **********************************
Input:   rs_ro_ratio - Rs divided by Ro
         gas_id      - target gas type
Output:  ppm of the target gas
Remarks: This function passes different curves to the MQGetPercentage function which
         calculates the ppm (parts per million) of the target gas.
************************************************************************************/
int MQGetGasPercentage(float rs_ro_ratio, int gas_id)
{
  if ( gas_id == GAS_LPG ) {
     return MQGetPercentage(rs_ro_ratio,LPGCurve);
  } else if ( gas_id == GAS_CO ) {
     return MQGetPercentage(rs_ro_ratio,COCurve);
  } else if ( gas_id == GAS_SMOKE ) {
     return MQGetPercentage(rs_ro_ratio,SmokeCurve);
  }

  return 0;
}

/***************************  MQGetPercentage ********************************
Input:   rs_ro_ratio - Rs divided by Ro
         pcurve      - pointer to the curve of the target gas
Output:  ppm of the target gas
Remarks: By using the slope and a point of the line. The x(logarithmic value of ppm)
         of the line could be derived if y(rs_ro_ratio) is provided. As it is a
         logarithmic coordinate, power of 10 is used to convert the result to non-logarithmic
         value.
************************************************************************************/
int  MQGetPercentage(float rs_ro_ratio, float *pcurve)
{
  return (pow(10,( ((log(rs_ro_ratio)-pcurve[1])/pcurve[2]) + pcurve[0])));
}
```

Then run the program (tip - it will take a while as the program will have to calibrate)

DHT11:

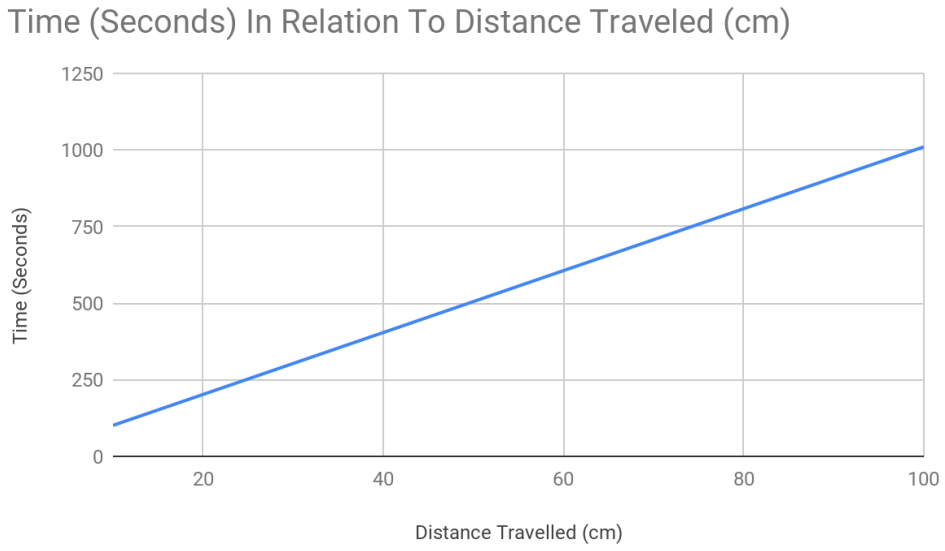
Connect the positive pin to the 3.3V pin on the Raspberry Pi (pin 1)
Connect Data to GPIo17 on the Raspberry Pi (pin 11)
Connect the negative pin to the ground pin on the Raspberry Pi (6)
Go to github.com/szazo/DHT11_Python
Click on the clone or download copy the link and paste it onto SSH terminal after typing out “git clone”.
Next type “ sudo nano dht11_example.py”
Then edit the code “pin = 17”
Then ctrl+x to save the file and press enter
The type “python dht11_example.py” and then the values will be displayed.

Data

Submitted by video

Analysis

From testing out our robot we can infer that it can perform all the pegs move vertically and horizontally, our claw moves up, down, and is able to grasp onto solid materials i.e. rocks, and our linear motion system moves our legs forward and back. Although we could improve on our claw as it can not pick up squishy objects. Also, we could make the bin bigger so it is easier to put objects in it. The claw could move 360 degrees and stoppers would be placed on both ends of the pegs. We continue to test what sorts of objects we can pick can pick-up and move around.



Conclusion

We can conclude the robot can move, collect samples, and collect data about gases around to an extent acceptance by our standards. Although we could put more sensors and scientific instruments to collect more data, as well as adding a big collection bin, and increasing the sturdiness of the robot by adding and connecting pegs together. If needed we could create a bigger claw to collect bigger samples.

Sources Of Error

-Wrong wiring - If you make a mistake in connecting the wiring of the sensors or connecting the motors to certain ports in the brain you will have a robot that doesn't work properly.
-Mistakes in Code - You must program your motors to respond to your remote control, and if that is incorrect your commands might give your different responses on your robot than you wanted.
-Incompatible parts - All your parts must be compatible with one another. For us when we were using VEX parts for the robot we had to use Vex brains, Vex batteries, and Vex remote controls. Yet for our sensors we could use a breadboard and attach it a raspberry Pi.

Acknowledgements

- Mr. Said - He answered a lot of our questions about the sensors.
- Mr. Landry - He supplied with VEX, arduino and raspberry Pi parts to create our prototype of the robot. He also gave us advice and tips to create a stronger robot. He also helped us on our VEX code.
- Ms. Dawes - answered questions about what data was needed on the logbook, and poster.

Future Directions Of Project

To continue in this direction would mean to create a fully-functional version of our model, then test and correct any imperfections found in the robot. As an end goal we hope to start production of our robot. One such imperfection is the sturdiness of the central pegs. We hope to connect the four pegs together, so they support each other and act as one. We can sell our robot to companies and use the profits to further advance upon our design and the technologies that are incorporated within it.

Application

This robot would allow scientists to collect samples (rocks) and data about volcanos without actually risking their lives venturing near one. This robot, unlike many others, would be able to climb near the top of the volcano with the traction provided by the grappling mechanisms. The large size of the robot would allow it to carry multiple pieces of scientific equipment. This robot would be able to monitor a volcano.

Model Specs

Robot

- Height - 34.25 cm
- Weight - 68.1326982245093333333333 Kg
- Length - 62 cm
- Width - 45 cm

Titanium

- Boiling Point - 3,287 °C
- Melting Point - 1,725 °C
- Density - 4.506 g/cm³
- Price per kg - $76.09
- Tensile Strength - 434 MPa
- 33.2 lb

Aerogel (Ceramic)

- $68.03/g
- Resistance to heat - 1400 °C
- Amount needed for robot (lb)

Motors

- Torque - 28.2 kg
- RPM - 20
- Horsepower - 2.00
- Amps 1.14 * (75 Motors)
- Watts - 2 watts
- Voltage - 12V
- Dimensions - 8.1 cm x 5.1 cm x 3.1 cm
- Weight - 181 grams
- Weight in titanium: 169.8553989466667 grams

Batteries

- 62.7 - ∞
- 12 V
- It would be located beneath the storage compartment

Sensors

- Hydrogen Sulfide
- Carbon Dioxide
- Humidity
- Temperature
- Sulfur Dioxide
- Carbon Monoxide

Radio Waves Transmitter Link:

- Height: 4.2 cm
- Length: 26.8 cm

- Width: 10.6 cm
- Price: $1461.06

Time of Flight Sensors
- Price: $5729.64 (1432.41 * 4)

Camera
- RPi NoIR Camera V2, Official Raspberry Pi Infrared Night Vision Camera Module V2.1 Sony IMX219 8-megapixel sensor 3280 × 2464 1080p30 for Raspberry Pi 3 2 Model B B+ @XYG
- Length: 30.635 cm
- Width: 1.5 cm

Raspberry Pi
- Model: Raspberry Pi 3 Model B+

Inner ear
- Back legs lift into ground more than front legs so it can climb on an angle

Remote
- What sort of remote would control this thing

Claw:
- Height: 7.5cm
- Weight: 817.4g
- Length: 21cm
- Width: 8cm
- It can lift 55.7 kg

O-Channel
- Height: 31.75 cm
- Length: 10 cm
- Weight: 0.4578096 kg
- Thickness: 0.32 cm
- Width: 11 cm

Pegs (Thing that lifts robot)
- Height: 31.75 cm
- Length: 9 cm
- Weight: 1.356032 kg
- Width: 2 cm
- Each peg can lift 56.4 kg

Pegs (Things that grapple into ground)
- Height: 31.75
- Weight: 1.458895756 kg
- Length: 9 cm
- Width: 2 cm

Linear slide
- Height: 2 cm
- Length: 30 cm
- Weight:1.08144
- Width: 4 cm
- Strength
- Length of Track: 61.4 cm
- Size of gears

Rack and pinion (thing on linear slide)

- Height 2 cm
- Length: 6 cm
- Weight: 1.3560327 kg
- Width: 3 cm
- Stall Torque: 28.2 kg (When attached to Motor)

Price: $ 12,475.0486324
Titanium
Motors
Raspberry PI 3B+
TeraRanger 3D Array *4

Raspberry Pi Infrared Night Vision Camera

Radio frequency transmitter (RC) and receiver

Bibliography

https://volcanoes.usgs.gov/vhp/gas.html
https://volcanoes.usgs.gov/vhp/gas_types.html
https://www.amesweb.info/Materials/Density_of_Titanium.aspx
https://agmetalminer.com/metal-prices/titanium/
https://www.youtube.com/watch?v=KUr8WgSIsfk&t=270s
https://www.youtube.com/watch?v=DPywqNps29E
https://playground.arduino.cc/Main/MQGasSensors
https://www.youtube.com/watch?v=Pv-WjVPJGMg
https://www.youtube.com/watch?v=JjPsW-7FUng&t=170s
https://www.instructables.com/id/ARDUINO-With-Ultrasonic-Sensor-distance-Measuremen/
https://www.youtube.com/watch?v=T8T6S5eFpqE&t=225s
https://en.wikipedia.org/wiki/Titanium
http://wiring.org.co/learning/basics/airqualitymq135.html