Exploring Air-Cushion Vehicle

PART A

A. *Abstract*

[1] The UGV that should be utilized for the avalanche hit condition should be fast, efficient and reliable. ACV(Air Cushion Vehicles) works on the basic principles of reaction force(Newton's third law of motion) caused by the thrust from air. We have handled every situation that the hovercraft might face in details and are confident about the project.

The regions where we intend to use this model are arid and uneven - thus handling it might become a major issue. Once the direct line of sight is lost, controlling the machine will be very difficult. We introduce the concept of PID control with Reinforcement learning to handle stability and Q learning based approach for manoeuvring (**Assistive Driving**).

The UGV will have 2 modes one will be manual and the other will be automated in which it will independently make decision to maintain the course of motion and stability, based on a number of sensor attached to it -which are described in details in a later sections. This will not only make the motion of the UGV stable but also allow the controller to use the machine with mere GPS signals and other sensors if required. Thus making the proposed UGV easy to use and decrease time in training the soldiers to use the UGV.

B. *Questionnaire*

1) Motivation behind participation: Every year many people have to suffer serious injuries due to avalanches and areas with loose snow.The major reason of them facing issues like frost bite and hypothermia is that today's transportation techniques aren't fast enough to rescue them. Loose snow is one of the few areas in transportation, where success in proper land mobility hasn′t been attained. We propose a ACV (Air Cushion Vehicle) to tackle the problem. Thus, saving lives not only of our mighty soldiers but also of local civilians in disaster hit regions.

Two major aims of our project is to ensure mobility in loose snow and extraction from snow. we propose a hybrid of a hovercraft and a snowmobile. this hybridization will help us make grip when required over the snow during mobility and allow a backward motion. The complete model is described in details in later section.

2) Specialized knowledge and expertise: Fluid Mechanics, Aerodynamics, Machine Learning and Q -Learning

3) Previous participation/awards/recognition: —

4) What are we planning to exhibit?: Innovative ideas, Concept/technologies depicted through 2D model, Prototype.

[1]DRUSE-TH02-1339

PART B

C. Physics Part

1) Working: The working of the ACV is based on simple but smartly applied physics- Newton's Third law, i.e., every action has an equal and opposite reaction.It was invented by Christopher Cockerell(who gave it a name ”Hovercraft”) in 1959. When a high speed channel of air is struck against a barrier(ground or water here a layer of snow), the speed of the air column decreases substantially in a short interval of time(impulse). This leads to a reaction force responsible for the uplift of the object to which the blower is attached.Additionally to cause displacement on the horizontal surface the vehicle is mounted with a specially designed fan which rotates to provide forward thrust.There are two popular ways to generating lift and thrust. One simple way is of using two separate propellers for thrust and lift. The other is quite interesting as it uses single power source to generate both lift and thrust. In this paper we have the second model as the reference for our UGV.
$Apictureofhovercraft$

A part of thrust from the fan(propeller) is used to fill the air in the skirt thus, developing the required uplift. In a more technical way, the whole vehicle consists of a board mounted on an air cushion. This air cushion overcomes the friction and the vehicle is now much more ”free” to move around propelled by the fan on its tail.

The air is forced into the hull cavity by the propeller through the holes cut out on the board, assisted by the incline. The air now gets filled into the hull chamber/cavity. Note that this stream of air has high kinetic energy. Holes are cut into the hull cavity and each hole is covered using a flexible and durable polymer(skirt). These skirts have a opening a at their bottoms from where the high velocity air stream forces its way out. In process it creates a cushion of air beneath the hovercraft. In this way the vehicle ”floats” on air. Further the navigation part of the hovercraft is controlled by the rudders and caterpillar wheel mechanism.

The vehicle is also equipped with caterpillar track mounted on three wheels, this makes this UGV a hybrid of ACV and a snowmobile. Thus providing us with best of both worlds, the wheels of caterpillar track are mounted on hydraulic arms, which are lowered in special condition like during extraction from snow and a better grip over the motion of the vehicle. The whole caterpillar track assembly is made out of reinforced plastic.

D. Material selection

When it comes to selecting materials for different parts we have to be really cautious. For each part there are different set of factors that are to be considered, which are solely decided be their specific purpose. Additionally several compromises have to be made with different parts mass, shape and material so as to satisfy the purpose of the vehicle as a whole. An additional factor that we have kept in mind is that since this UGV has the purpose of saving the lives of our brave soldiers hence we have tried to focus more on factor of safety and using more advanced technology rather than cutting cost.

*1) **Hull:*** Factors that we have considered are strength, weight, durability,corrosion and stiffness.
Strength – The ability of the material to support a load without breaking (physical failure)
Stiffness – The ability of the material to distribute a load and resist deformation or deflection (functional failure)

S.No	Parameters	Values
Material	Young's modulus	Density
Steel	200 GPa	7.7 g/cm^3
Al	69 GPa	2.7 g/cm^3
Carbon Fibre	500 -1000GPa	1.59-1.7 g/cm^3

Note: Kevlar is another material that we considered. Its Young's modulus is higher than certain grades of carbon fiber. But the problem with Kevlar is that its ability to withstand compressive load is much lower than carbon fiber.

Considering the table shown above and the current popularity in the automobile industry we chose carbon fiber to be out hull material.

*2) **Deciding thickness**:* I have applied the case of beam buckling in order to decide the thickness of the CF board used to construct the hull.
Using Euler's buckling,
$P_{cr} = \pi^2 EI/k^2L^2 =$
$= 739.5N.$
The critical load is 739.5 N if we consider the thickness of each CF sheet to be 6 mm. Assuming that the total weight of the vehicle is equally distributed among the four sides. Each side has to support an approximate weight of 250N.
Therefore, $FOS = 739.5/250$
$= 2.96$
calculation of mass
Here also we went for a fair estimation process to calculate the upper limit hence in the final design we will get some advantage on payload. $Mass of the upper board = density times the volume used.$
$= 1.7 * (0.006 * 1.5 * 0.75)$
$= 11.4kg$
$Mass of lower board = 10kg.$
Total mass of the hovercraft is = 21.4 kg.

*3) **Duct**:* Duct is a part where we can slightly compromise with cost and strength and simultaneously decreasing the weight significantly. This time we compare among

- Carbon fibre
- Aluminium alloy
- PVC
- ABS material

Carbon fibre is lighter than aluminium (heavier than PVC and ABS)but the strongest of all(in fact too strong and costly for our business here). PVC cheaper and strong enough. ABS is lightest of all with a density of about 1 gm/cm^3 and is as strong as PVC. Hence this time we go with ABS material for our duct. It has got Yield stress of 44 MPa.
$Mass of the duct = 9.6kg.$
To provide an aerodynamic advantage to the vehicle we modified the structure of the duct(only significant when the speed is high).

*4) **Motor**:* We use motor to control the motion of the rudders, rotate the propeller and drive the caterpillar wheels.
$Motor for the rudder weighs 2.7kg.$
$Prop motor weighs 10kg.$
$Total mass = 12.7kg.$

*5) **Propeller**:* Here we went with a standard choice of aluminium alloy for propeller material. This choice strong enough and is corrosion resistant. It has a weight of 3 kg.

PART	MASS (kg)	X- POSITION(cm) From centre	Z–POSITION
Hull	21.4	10.5	10
Rudder motor	3	73	25
Prop motor	10	25	57.5
Duct	9.6	45.5	57.5
Propeller	3 - 5	27	57.5
Rudder	3	73	55
ET	15+5(motor)	centre of mass after calculation	–

ET = Emergency Traction
Z–POSITION =(height from the ground in cm) [2]

E. *Mobility and lift*

To lift the payload and to support its own weight it would require a power source. This power will be used to push air beneath the craft and lift it. This power is given by following equation

$$\text{Power} = \text{Cushion Pressure} * \text{Volume flow rate out of the skirt}$$

When the hovercraft lies on the ground(in switch off mode) the pressure exerted on the ground is weight of the craft upon the base area. So in order to lift the craft on an air cushion the air cushion pressure must be equal to that in the previous case when the craft was lying idle on the ground. Therefore, mathematically,

$$\text{CP}= \text{Weight of the ACV/Area of the base}$$

Deciding dimension of the craft
First things first; the aim is to lift a payload of around 30 kg on the board and transport it during the mission. Lifting the weight with decent power required is the major factor. To provide the lift and forward thrust we found that it requires a propeller of 0.75m diameter. Also minimise the material used the breadth is taken to be the same as the diameter of the propeller. Also from the data sheet of hovercraft engineering the length of the hovercraft is twice its breadth.
Since the length and breadth of the UGV have been decided beforehand hence, the base area can be determined. So for this chosen area sufficient air has to be blown beneath the vehicle to balance the total weight. For lift up we will be using **Momentum curtain theory** (a modified version of the open plenum theory) to provide an extra lift to the vehicle.

Momentum Curtain theory Revolutionised the way air is pumped beneath the craft. Previously air was directly pumped beneath the craft. This led to considerable leakage of air. Hence this method is quite inefficient. Instead of this we bring an inner container called the “curtain”(the area shaded black in the figure is the curtain). This substantially reduces the area through which air leaves now, avoiding any leakage. It also creates more pressure compared to open plenum(when no curtain is used) because the exit region acts like a Nozzle. The skirts further increase the peak pressure capability.

[2]DRUSE-TH02-1339

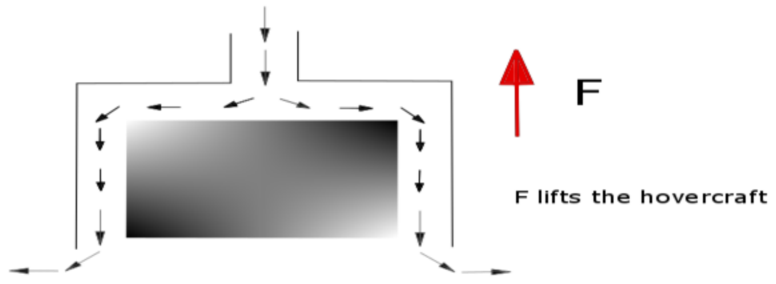


Fig. 1. Momentum curtain

S.No	Parameters	Values
1.	Hovercraft length	1.5 m
2.	Hovercraft width	0.75 m
3.	Estimate air gap, hg	2.54 cm
4.	Hull weight	21.4 kg at max
5.	Estimate pilot weight	0kg
6.	Weight of the sensors and other driving assisting devices	5kg
7.	Weight of the payload	20+5kg
8.	Total craft weight without payload	69kg
9.	Area of base	1.125 m^2
10.	Skirt height on complete inflation	20cm

P_e(air exit pressure) = $\frac{Wt(weight)}{A(BaseArea)} = 100/1.125*9.81 = 872N/m^2$

However for practical purposes we have to maintain a pressure 15 percent more than the calculated theoretical value. This is due to leakages, bends, compression and drag the happen in the process.

$(CP)' = 1002.8Pa$

Area of air gap, A_g = 0.0254*2*(1.5+0.75) = 0.1143 m^2

Velocity of escaping air,

$$V_e = \sqrt{\frac{2P_c}{\rho}}$$

So, V_e = 41.6 m/s

Flow rate = Q = A_g*V_e = 4.753 m^3/s

A typical propeller efficiency, η_f = 0.85

Required engine Power = 4.7 KW

Note that this is the same volume that is being forced into the hull by the propeller and then expelled out through the skirts.

So intercepted volume rate = 4.753 m^3/s

Now, the area of the propeller that is blocked is half of the total area of the duct.

$Areaoftheduct = \pi * d^2/4$

$= 0.442$ $Interceptedarea = areaofduct/2$

$= 0.221m^2$

Now applying equation of continuity at the propeller section

$Volumeflowrate = Vppld * Ainstpd$

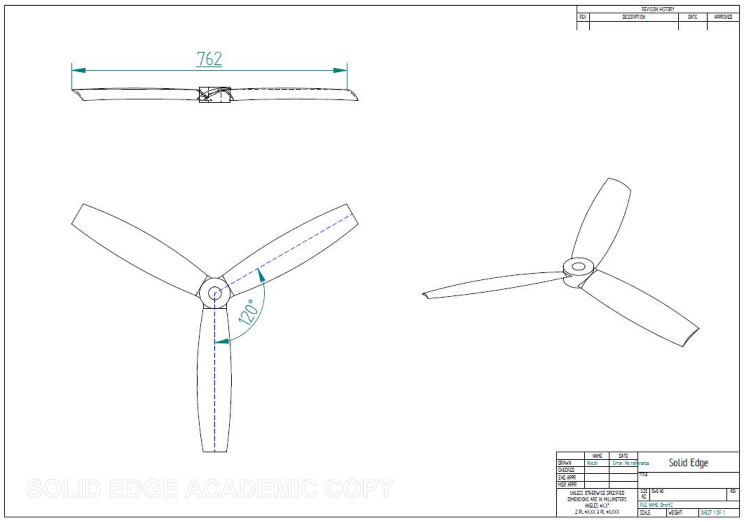
$Vppld = 21.51m/s$

From the simple momentum equation we can write

$Thrustgenerated = ro * Afree * Vppld^2 = 118.6N$

Since the propellers are not 100 percent efficient therefore we use a propeller with 85 percentage efficiency. Also the amount of thrust generated decreases for a given propeller rpm as the craft gains speed.
Hence,
$T' = 136.39N$
We use the empirical results that have obtained by a researcher Gabriel Staples that relates thrust, diameter of propeller, pitch of the propeller and the RPM of propeller.
formula comes in here
We get required RPM = 4000 for pitch = 10 in
We get required RPM = 3460 for pitch = 20 in
Therefore we chose pitch = 20 in.

2. **Forward thrust**
A specially designed fan must be used to provide the forward momentum to the hovercraft. These fans are called propellers.
Propellers- we plan to use use 3 blade propeller to drive the ACV. This number is chosen to be 3 for the sake of compactness of the ACV meanwhile getting the same thrust.



Propellers are nothing but just rotating wings. Due to a distribution in velocity along the radial direction the lift generated is not constant in common blades. Hence, the blades we intend i.e. Bullnose Propeller are designed such that the angle of attack decreases radially. This in turn compensates for the lower lift force at the root and increases the amount of thrust at the root.

Increasing the propeller-efficiency(DUCT THEORY) From the principle of aerofoil we know that there is a difference in pressure between the sides of the propeller(asymmetric aerofoil profile). Due to this reason something different happens on the periphery of the blades. The air starts to flow from high pressure side to low pressure side; this leads to formation of vortices - which are sources of dissipation and cause fall in efficiency.At the vortices the sir starts to form ”swirling” pattern, which drains the energy(kinetic energy)from the system. Construction of a duct(barrier) around the blades avoids the cross flow and reduces the total power required. Moreover, the periphery of the duct can be stretched outwards radially(creating yet another “aerofoil” that assists the thrust).

The first challenge to overcome is friction. Although negligible because of the uplift, friction has to be taken in account when we scale up the model. To measure the friction force being applied on the craft, when it tries to move can be estimated using the classic experiment of spring balance.

The craft is attached to a pulling rope which in turn is attached to a spring balance and then the set-up is pulled. Initially the friction (after the craft achieves the desired lift) opposes and reading can be taken from the spring balance. The reading of the spring balance just before the craft moves will give us the value of static friction (f_s). Finally when the craft is set into uniform motion we get slightly reduced value of the resisting force kinetic friction(f_k). So now if an equal amount of force is applied to the craft and then given a slight push, it would start to hover.

Power from engine required to overcome friction,

$$P = fs * vo$$

However this is not over, not yet. We will require to accelerate the craft which would require even more amount of thrust,

Suppose V is its top speed (10kmph) then the maximum power of the engine can be approximately estimated using this speed in the above equation. This top velocity will help in the estimation of total final power.

A typical value of friction coefficient over snow can be 0.1-0.4 and since there is almost no contact with the ground, we can say that the effective normal reaction that is responsible for friction is 1/ 10 of the total normal reaction (65 newtons).This is the maximum friction condition for normal reaction is less than the 1/10 of the weight.

$$\text{Force} = \mu*(65/10)*g = 0.1*6.5*10=6.5N$$

Hence in this case the force turns out to be at max 6.5 N (given the total mass here is 65kg which shows that friction is really small.)

Additionally, consider the drag force(resistive force dependent on the velocity) as well. And using the expression for resistive drag force, we can arrive at an estimated value of

$$\text{Resistive Force, } R = \frac{1}{2}\rho v^2 A * C$$

ρ density of the regional air

v speed at which the craft is moving

A projected area of the front end of the craft

C Coefficient of drag

The coefficient of drag depends on the size, shape texture etc. of the body. This is determined experimentally using *wind tunnels or theoretically using the simulation softwares*. Additionally the coefficient of drag for the auto-mobile range from 0.4 for cars in 1980s to 0.2 in modern day cars. Even lower coefficient of drag is possible. And for hovercraft application this can be taken to be 0.4 (Because of the half tear drop design). Hence it turns out that for a speed of 10 km/hr the drag force is around 2 N. [3]

$$\text{Hence, } PowerRequired = (F + R) * v$$

$$[v=(v_o+at), \text{ if acceleration is constant}]$$

Hence the total power required = (Friction + R + Accelerating force)*v when the hovercraft is moving at an instantaneous speed of v. [v=(v_o+at), if acceleration is constant]

$$P = (10)*3 \text{ [since max velocity can be 3m/s]}$$
$$= 30 \text{ W. (without any acceleration)}$$

[3]*Note: Increasing the propeller-efficiency(DUCT THEORY) From the principle of aerofoil we know that there is a difference in pressure between the sides of the propeller. Due to this reason something different happens on the periphery of the blades. The air starts to flow from high pressure side to low pressure side; this leads to formation of vortices - which are sources of dissipation and cause fall in efficiency. Construction of a duct(barrier) around the blades avoids the cross flow and reduces the power required. Moreover, the periphery of the duct can be stretched outwards(creating yet another “aerofoil” that assists the thrust). **trapezium - Image**

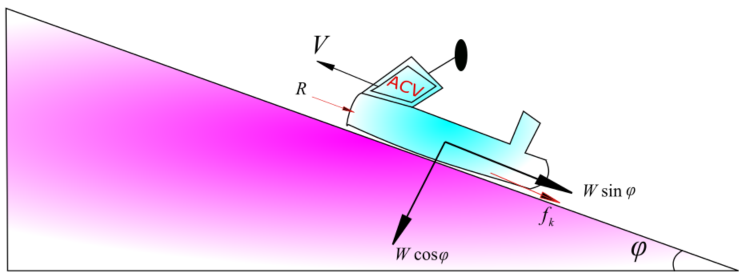


Fig. 2. Handling Slope

Moreover, the power efficiency can be increased by using the a duct around the fans(by simply avoiding the energy being wasted in the form of vortices*).

Note: Power-to-weight ratio is equal to thrust per unit mass multiplied by the velocity of any vehicle.

1) Managing Slope : [4]

To maintain a particular speed on a slope we have to consider the fact that, what could be the slope of the terrain or the mountain that the craft will be hovering on.On an average the avalanches occur on the mountains with slopes around 30^o. Hence, at max it has to climb on a slope of say 30^o. The required power to maintain a constant velocity v_o would be

$$power = (f_s + W sin(\phi) + R) * V_o \text{ -With usual notations}$$

Maximum power to be supplied by the driving engine for a given velocity occurs when the mountain is really steep say$(\phi = 30^o)$

So to overcome the resistive force on ground we would need a power of about 30 watt. Moreover, the power efficiency can be increased by using the a duct around the fans(by simply avoiding the energy being wasted in the form of vortices).

It might seem that power requirement has been substantially increased due to the $m * g * sin(\phi)$ component. But that is not the complete picture. Of course the net requirement of power in the horizontal direction has substantially increased meanwhile the power to lift has substantially. So the net power requirement does not vary much.
Additionally, to maintain stability of the craft on such inclined terrain the velocity is restricted to 1 m/s. Hence the new and the maximum power requirement is
P= (2+6.5+100*9.81)/2
=495 watt.

Hence it can be concluded that the maximum power required is 3.8+0.5 = 4.3HP. And normal working power is 3 HP. So we require an engine that can give power output of about 4.5 HP.

2) ***Energy Consumption - Go green:*** The table below describes the power requirement by the UGV in detail:

[4]DRUSE-TH02-1339

S.No	device	Consumption
1.	Servos	(approx.)5W * 14
2.	Actuators	(12V,5A)*6 = 360W
3.	Motor(For Propellers)	500W
4.	Radio Transmitter	(12V,1A) * 2 =24W
5.	Raspberry Pi	(5V,2A)* = 10W
5.	Arduino	(9V,45mA) *3 = 3W

Total power requirement sums up to be 980W
For running the machine for 1 hr, we will require 980Wh of energy. Possible configuration of battery will be (12V,80Ah)

*3) **Skirt**:* Segmented skirt.

The defining component of hovercraft that enables it to rise above the ground.When the hovercraft was initially built no skirts were used. This required quite large amount of power to be consumed in the lifting process. The was that the craft was not able to maintain sufficient pressure beneath. This is where the skirts fit in. They maintain the pressure beneath the vehicle. and maintain a constant and stable hovering height.
Some factors to be kept in mind while deciding the nature and type of skirt

- Considerable amount of elasticity
- Should be able to withstand large amount of energy from collisions
- Minimize loss of cushion air

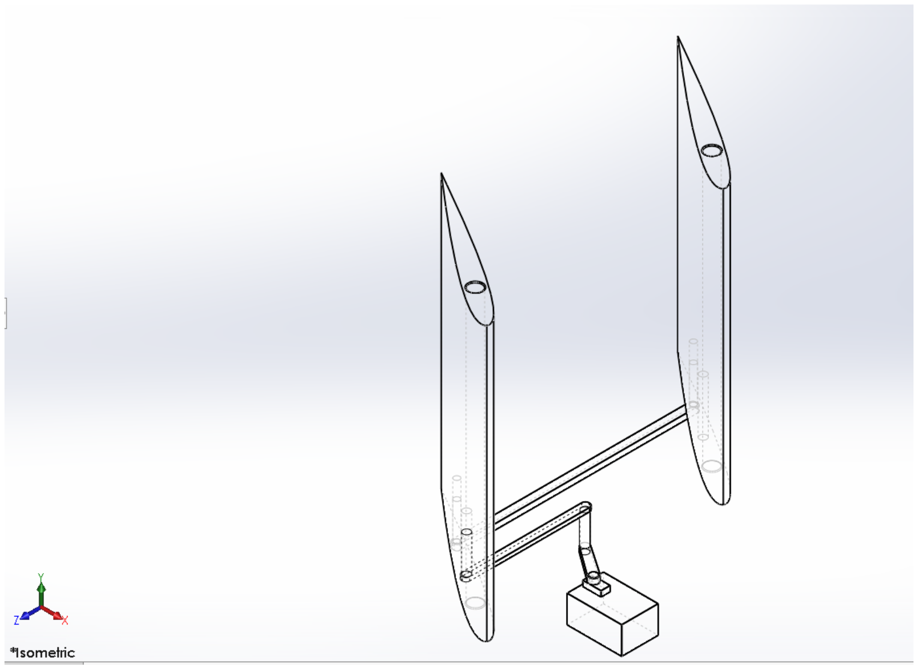
A generally used material is Neoprene coated nylon.

We intend to use segmented skirt because of it$'$s feasibility, as in case of damage we will only have to remove a segment of the skirt rather than the whole skirt.

*F. **Rudders***

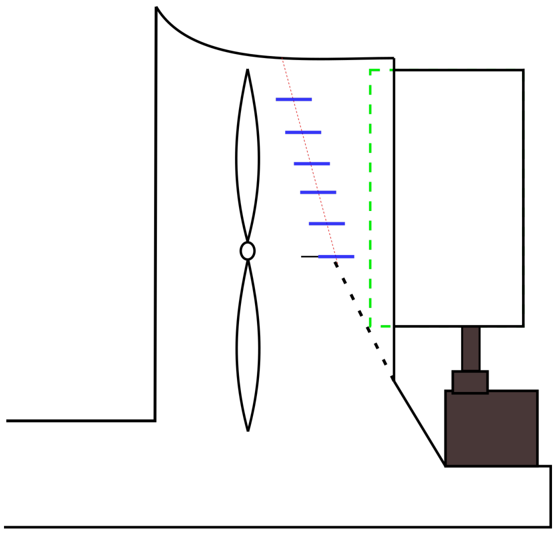
The rudders are the soul of this craft. This is the part that will be responsible for maintaining the path for the targeted mission. Any change in direction is not possible without them.

*1) **Working and constructions.**:* They are two in number and are attached at the rear of the duct with a small part of it inside the duct.Each is a symmetrical aerofoil.
This assembly is operated using a servo motor. This mechanism uses a 4-bar mechanism in order to attain the desirable motion of the rudder and hence the hovercraft. At first site it might seem that this mechanism is a 6 bar mechanism. However on close enquiry it can be found that the link attached to the motor and the next link act as dummy link. Link 5 has same angle from the ground as the link 2 and link 4. Also the link 6 is always parallel to the link 3. The link 5 is attached to servo motor.
We intend to use a servo for its high torque supplying ability, its precise movement and easy controls. It can be noted that the degree of freedom of the mechanism is one. The maximum moment is required when the angle between the rudder and the air flow is 90 degree(due to direct obstruction). Further upon calculation it is found that the required supporting moment at the driving joint (where the servo is present) in this condition is 18 N-m. We intend to use carbon fibre as the rudder material owing to its strength and light weight.



Isometric View of rudder mechanism

G. ***Flaps***

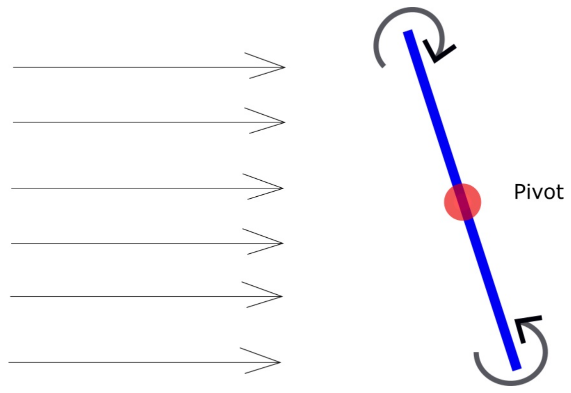


Rudder and flap mechanism

Above is the schematic diagram of rear of the vehicle. The blue marked lines show the “flaps”. These flaps are the airflow directors.

*1) **Purpose:*** It directs the airflow. When in horizontal position it allows the air to flow through and the hovercraft moves forward due to the thrust generated. However when the flaps are all aligned along the red line they block the airflow. Further due to its inclined design it facilitates that volume of air to go beneath the vehicle. The reopening due to air pressure is restricted by design.

*2) **Design:***



Flap mechanism

It is noteworthy that there is no net torque that is acting on the flap. Hence making it quite easy to rotate the flap about the pivot(marked in red). We intend to use small servo motors for each such flap that will enable its opening and closing. These servos are attached to the outer side of the duct. It should be noted the servo is attached to the tip of the flap not the pivot. This is done to maximize the torque applied by the servo. Each flap covers has breadth of 10cm, thickness of 0.2cm and covers the part of duct.

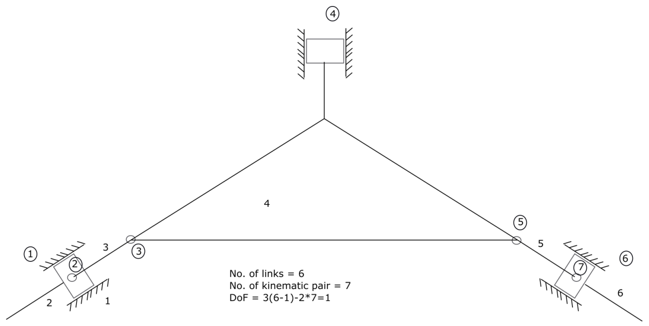
*3) **Toppling : A rare case**:* As the centre of gravity of the ACV is very low, the chances of it getting toppled upside gets minimized. The PID controller will constantly look out for the cases where the hovercraft crosses a critical sensor rating and take action accordingly- More on this is described in a later section.

*H. **Self Extraction from snow***

Self extraction from snow is another important area to consider. Hilly areas are bound to have unpredictability. This gives rise to situation where the ACV might get stuck. So in that case we plan to use caterpillar wheel mechanism implemented using caterpillar tracks mounted on 3 wheels driven by one of the wheels(top one to be exact). So whenever the vehicle finds itself trapped or stuck, it deploys this mechanism by lowering the whole assembly.In the mean time it brings down the flaps such that no thrust is being generated. Now the vehicle gets the necessary added traction for forcing out of the snow. Reaching a stable position and then continuing on its path.

The ACV uses momentum curtain theory instead of open plenum(when the obstruction in figure is removed) to generate lift for the craft. This leave the hovercraft with a substantial amount of space at the centre. This space will be utilized to house the caterpillar track mechanism described above.

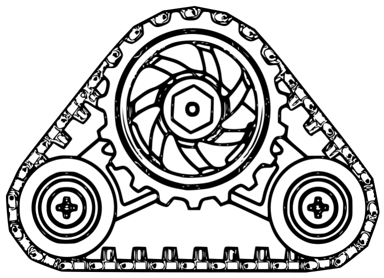
To take care of extraction we propose the use caterpillar track mounted rooler mechanism, these tracks are mounted on 3 axles, with 2 lower axles on the same level, and rest one a bit higher. Thus making a triangular structure. While reversing we intend to close both the air flow control lids making the air blow directly into the skirt, thus making sure to create a stable equilibrium position.



The Degree of freedom of this mechanism is 1 so it can move only in vertical direction

There will be 3 hydraulic arms attached to the roller mechanism, the axle on the top is the power axle which runs the the whole assembly, This power axle is driven by a stepper motor. One hydraulic arm is attached to each axle. The caterpillar track will be made up of CFRPG, with grooves, when the rollers roll we will get drag, which helps us to move in the specified direction.

Rollers are based on how snowmobile works, the ice stuck between the groove provides the necessary traction. Caterpillar track, is a system of vehicle propulsion in which a continuous band of treads or track plates is driven by two or more wheels- (here 3 wheels are used) as shown in the figure.



I. Connectivity

The UGV has to be controlled remotely from a large distance so we need to be aware of the surroundings, thus UGV is equipped with various sensors and GPS which will provide data necessary for the the person sitting at the base station to decide which direction to move4.

In order to control the UGV we must be able to communicate with it wirelessly. For that purpose we will use direct radio communication between the UGV and the control station.

For radio communication to work, we must have a transmitter, to send the messages and a receiver to get the messages. Also the movement of the UGV depends on the data received from the sensors, so we will have receiver as well as transmitter on both the ends. The transmitter and receiver at both ends will be tuned to the same frequency. These transmitters and receivers will be in the form of RPi900 module connected to the raspberry pi at both ends. This module operates in 900MHz band.

We have selected 900MHz frequency because lower frequencies tend to have a much greater range at lower power than higher frequency devices. They also have a greater ability to penetrate dense objects which is another reason why they are great for remote controlling of the UGV. Most remote control drones also use lower frequency like 900 MHz for transmission.

The various equipments on the UGV will collect the readings, then this data will be sent to the controller where decisions can be made accordingly by the controller (The whole process is explained in detail in the later sections).

J. Computation and Sensors

S.No	Sensors	Position	Usage
1.	IMU	Center of the vehicle(above the hull)	Orientation and acceleration
2.	GPS	can be placed anywhere	Geo Location
3.	Thermal camera	Front	For detecting warm bodies
4.	LIDAR	one at top and other at front	3-D image of the surroundings
5.	RADAR	Front	Object sensing

The UGV being an autonomous device will requires significant amount of computation power,Raspberry Pi is capable of doing the necessary amount of Computer vision related computation,thus can be used for this purpose, but for a smoother experience we suggest the use of more powerful processor alternatives like those used in self-driving cars like Intel go, NVidia drive etc.

*1) **Thermal camera sensor (AMG8833):*** This sensor connects to the Raspberry Pi via I2C protocol and provides a thermal image of the surroundings. This sensor measures temperatures ranging from 0 degree Celsius to 80 degree Celsius(32 degree Fahrenheit to 176 degree Fahrenheit) with an accuracy of $+/-$ 2.5 degree Celsius(4.5 degree Fahrenheit). It can detect worm bodies from a distance of up to 7 meters (23) feet

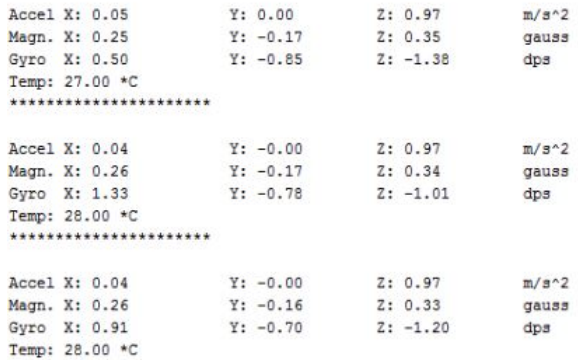
.

It has maximum frame rate of 10Hz(Frames per second). This sensor is an 8x8 array of IR thermal sensors so it returns an array 64 individual temperature readings over I2C protocol. Appropriate colours can be decided for different temperatures thus providing an alert at a certain temperature range or we can also use this data to generate a thermal image of surroundings.

Sample code that returns an 8x8 array of temperature readings: Thermaltest.py (Appendix 1)
Sample code that makes a thermal image of surrounding using pygame: Thermalcam.py (Appendix 2)
(Note: these codes are a part of official Adafruit repository of AMG8833 sensor on GitHub modified for our use)
This sensor can be used to detect warm bodies, this bodies might be humans. This sensor is one of the important component of the UGV when it comes to finding humans.

*2) **Accelerometer, Gyroscope, Magnetometer(LSM9DS0 - IMU(Inertial Measurement Unit):*** The chip contains three sensors, one is a classic 3-axis accelerometer, which can tell which direction is down towards the Earth (by measuring gravity) or how fast our UGV is accelerating in 3D space. The second is a 3-axis magnetometer that can sense where the strongest magnetic force is coming from (Helping us to get a sense of direction), generally used to detect magnetic north. The third is a 3-axis gyroscope that can measure spin or twist. By combining this data, we can know the orientation of our Hovercraft.
The LSM9DS0 has a linear acceleration full scale of (g = acceleration due to gravity) ± 2g/ ± 4g/ ± 6g/ ± 8g/ ± 16g, a magnetic field full scale of ± 2/ ± 4/ ± 8/ ± 12 in gauss and an angular rate of ± 245/ ± 500/± 2000 dps (degrees per second). It also connects to the raspberry pi via I2C protocol.

This sensor gives the readings in a very simple format and using the given sample code, the sensor data can be printed in the following format.



```
Accel X: 0.05        Y: 0.00        Z: 0.97        m/s^2
Magn. X: 0.25        Y: -0.17       Z: 0.35        gauss
Gyro  X: 0.50        Y: -0.85       Z: -1.38       dps
Temp: 27.00 *C
**********************

Accel X: 0.04        Y: -0.00       Z: 0.97        m/s^2
Magn. X: 0.26        Y: -0.17       Z: 0.34        gauss
Gyro  X: 1.33        Y: -0.78       Z: -1.01       dps
Temp: 28.00 *C
**********************

Accel X: 0.04        Y: -0.00       Z: 0.97        m/s^2
Magn. X: 0.26        Y: -0.16       Z: 0.33        gauss
Gyro  X: 0.91        Y: -0.70       Z: -1.20       dps
Temp: 28.00 *C
```

As the data received from the IMU may contain many errors, keeping in mind the possible errors from these sensors, the UGV is thus equipped with more than one such sensor(Here 2) . It uses sensor fusion technique to get a mean of the readings of all the sensors so as to improve our accuracy.
Sensor fusion is a technique that intelligently combines data from several sensors for improving application or system performance. Fur further improvement in accuracy different type of filters can also be used.
The biggest issue with using two completely stand alone systems(IMUs) lies in synchronizing the system clocks to get time sensitive information - the IMU reading from the two sensors need to match up exactly so that they can be combined to obtain a better estimate. The sensor values are considered only when the time stamps on the messages match. A margin of 0.05 seconds is allowed to ensure that important IMU events were not lost in case the time stamps did not match exactly.

*3) **GPS Module(LS20031/SUP500F) :*** The LS20031 GPS receiver is a complete GPS antenna receiver that includes an embedded antenna and GPS receiver circuits. The receiver is based on the proven technology found in LOCOSYS 66 channel GPS SMD(Surface-mount device) type receivers that use MediaTek chip solution. The GPS smart antenna will track up to 66 satellites at a time while providing fast time-to-first-fix, one-second navigation update and low power consumption. It provides an accuracy of about 3m (2D RMS).
For navigation, we can use Navit, which is an open source navigation system with GPS tracking. It uses serial communication on RX/TX to connect t raspberry pi. This GPS gives output in the form of NMEA strings. These strings contain different GPS data, such as latitude, longitude, ground-speed, altitude, etc. For our project, we needed latitude, longitude and atitude which are present in multiple NMEA strings, including the RMC that is outputted five times per second in the default configuration.

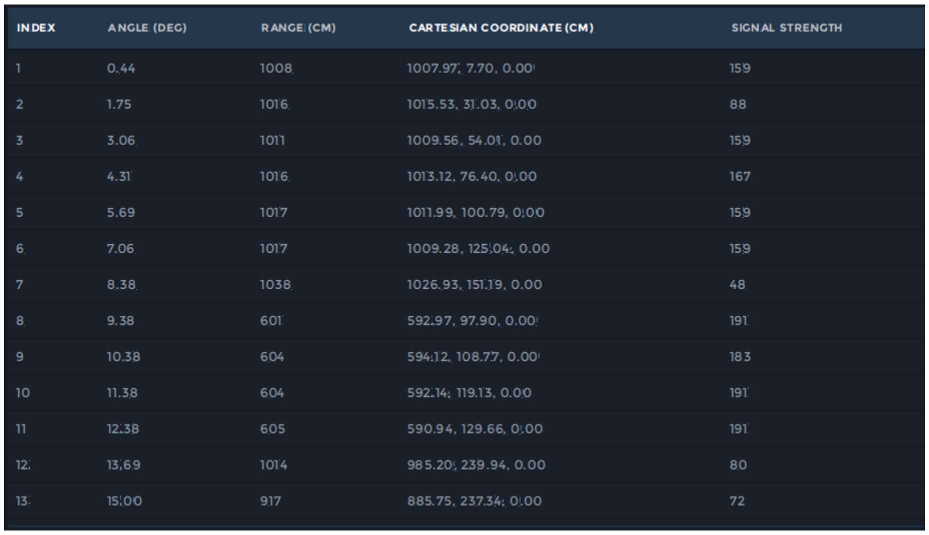
To read the GPS device and select RMC strings we used the following code: rmcread.py (Appendix 2) This RMC string has a defined format; we can extract the required data(Latitude and Longitude) from this string using the following code: getlatlong.py (Appendix 2) Using the latitude and longitude the location of UGV on the map can be determined. For the autonomous driving case this data will be used by the UGV to follow it's defined path. For the user-controlled case this data will enable the controller to determine the location of the UGV on the map.

*4) **LiDAR lite v3 - Scanse Sweep:*** It is a compact optical distance measurement sensor from Garmin. It features an edge-emitting, 905nm (1.3 watts), single-stripe laser transmitter, 4 mili Radian x 2 mili Radian beam divergence, and an optical aperture of 12.5mm. It has a range of 0-40m and provides an accuracy of +/-2.5cm at distances greater than 1m. It is user- configurable, allowing adjustment between accuracy, operating range and measurement time(Example: If we increase the time for measurement then the accuracy of readings can be improved).It is connected to raspberry pi via I2C or PWM. The distance is calculated by the formula

$$d = c(t/2)$$

where c is the speed of light and t is the time taken by light beam to return to the receiver. For our purpose, we use Scanse-Sweep, a LiDAR device that can be used directly by connecting it to the raspberry pi by Serial communication using Rx/Tx pins or USB through a UART connector. A LiDAR lite is housed inside the device, which rotates at user-defined speed (1Hzto 5Hz) to take 360 view of the surroundings. With the help of attached motor, we can rotate the sensor module at our desired speed; its range is between 2Hz to 10Hz(revolutions per second). We will be using two such LiDARS :
One for making a 2-D map of surrounding obstacles
The other to detect cliffs or valleys in the path The first LiDAR will be mounted vertically on the roof of the Hovercraft (at a height of (0.4m)and the second one will be mounted at some angle (let it be 45 from the vertical) at the front so that when it rotates it could detect cliffs and valleys in front of the vehicle and also detect small obstacles.

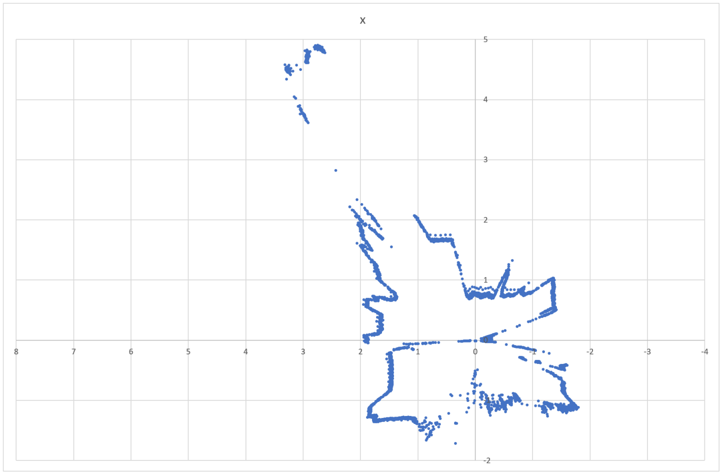
The data generated from the LiDAR will contain the distance of obstacles at each degree from a reference line and the strength of signal received. The strength of the signal depends on the distance of obstacle as well as the light absorbing nature of the obstacle.



INDEX	ANGLE (DEG)	RANGE (CM)	CARTESIAN COORDINATE (CM)	SIGNAL STRENGTH
1	0.44	1008	1007.97, 7.70, 0.00	159
2	1.75	1016	1015.53, 31.03, 0.00	88
3	3.06	1011	1009.56, 54.01, 0.00	159
4	4.31	1016	1013.12, 76.40, 0.00	167
5	5.69	1017	1011.99, 100.79, 0.00	159
6	7.06	1017	1009.28, 125.04, 0.00	159
7	8.38	1038	1026.93, 151.19, 0.00	48
8	9.38	601	592.97, 97.90, 0.00	191
9	10.38	604	594.12, 108.77, 0.00	183
10	11.38	604	592.14, 119.13, 0.00	191
11	12.38	605	590.94, 129.66, 0.00	191
12	13.69	1014	985.20, 239.94, 0.00	80
13	15.00	917	885.75, 237.34, 0.00	72

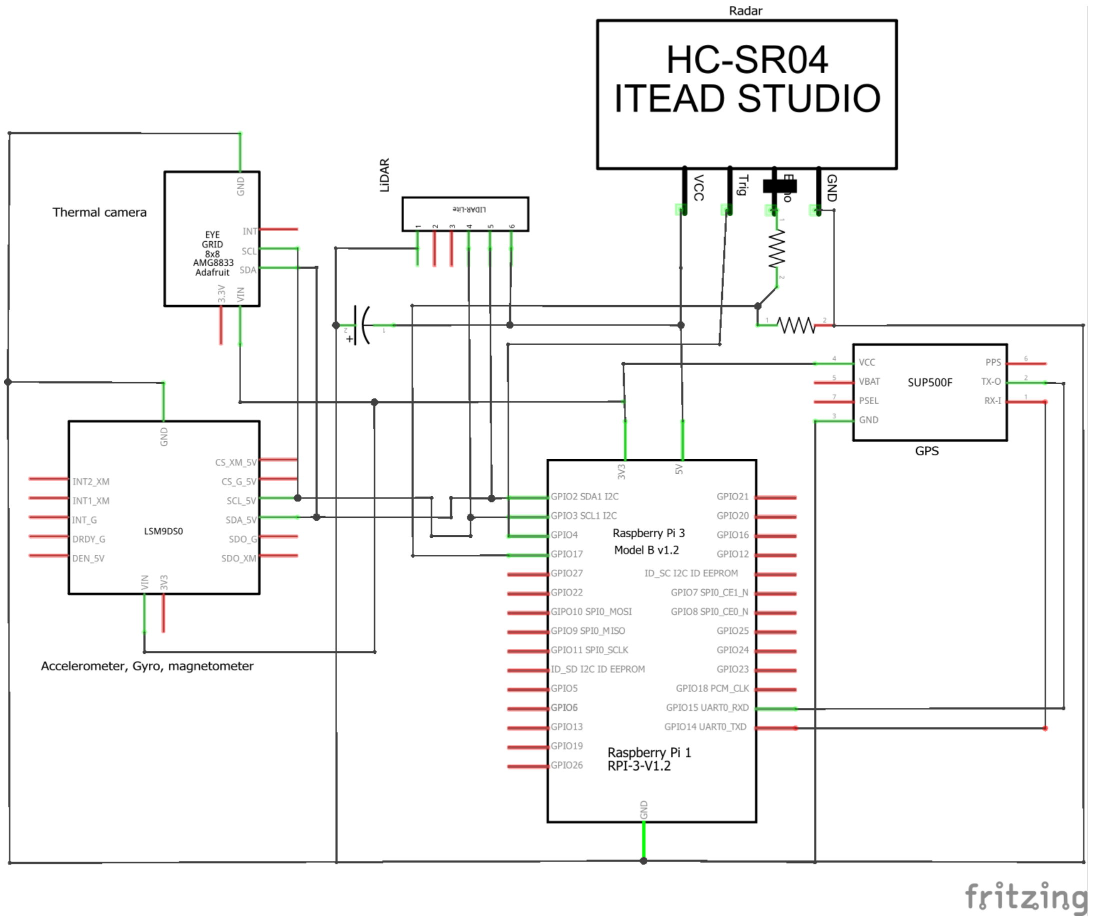
Sample data table for LiDAR readings

This data will be used in two ways :
First for the Controller based operation, this data will be sent to the controller where the data will be assembled and a 2-D graph of data will be created at the user end, which will show the distance of obstacles around the UGV. Thus, it will enable the user to decide the direction of movement.



This is a sample data plot from the readings received from the LiDAR, Here the LiDAR is at the origin, the blue points represent the obstacles, and the white space indicates the path. Similarly, for the second LiDAR this type of 2-D map will depict the depth of the path around the UGV. This whole operation will take 2-3 seconds to complete, as LiDAR will have to take more than one reading of the same point to increase accuracy.
Second for the Self-driving part, the data generated from the LiDAR will be saved in the form of a CSV file, and this file will be refreshed every second. Based on the readings in this CSV file, the machine-learning algorithm will be able to know the current state and predict the direction of motion of the UGV. Following is a sample code for taking the readings from the LiDAR and saving the readings in the form of a csv file: sweep.py (Appendix 6)

*5) **RADAR (HC-SR04) :*** This sensor provides the working of a radar. This sensor provides 2cm to 400cm of non-contact measurement functionality with aranging accuracy that can reach up to 3mm. It has a measure angle of 15 degree. Each HC-SR04 module includes an ultrasonic transmitter, a receiver and a control circuit. This sensor gives the distance of an obstacle in front of it. When the user is controlling the UGV this sensor will give an alert (on a high priority basis), if the UGV is constantly approaching an obstacle. When the UGV is in autonomous driving mode this sensor will generate an warninIt is noteworthy that there is no net torque that is acting on the flap. Hence making it quite easy to rotate the flap about the pivot(marked in red). We intend to use small servo motors for each such flap that will enable its opening and closing. These servos are attached to the outer side of the duct. It should be noted the servo is attached to the tip of the flap not the pivot. This is done to maximize the torque applied by the servo. Each flap covers has breadth of 10cm, thickness of 0.2cm and covers the part of duct.g to stop or slow down the motors in case of constantly approaching an obstacle. Sample code to print distance determined by HC-SR04: radar.py (Appendix 7)



Connection of various sensors on the UGV

K. Control And Steering

Sending data to the controller : After getting data value from all the sensors the data must be sent to the controller, to take appropriate action. To accomplish this task we are using RPi900.
RPi900 is an interface board for the Raspberry Pi single-board computer. Combined with a DNT900 radio transceiver from Murata, it provides a cheap, powerful solution for any project requiring long-range, wireless data transmission. The DNT900 is a 900 MHz, frequency-hopping spread-spectrum radio transceiver with maximum transmitting power of 1W and a range of up to 64.3km . Data transmission rates of up to 500 kb/s(Kilo Bits per second) are possible, depending on range.As we expect the UGV to be under a 5km radius it is possible to achieve 60 to 70 percent performance in data transfer. The radio also includes analog and digital I/O for sensor applications. It operates in the ISM(Industrial scientific medical) band and is FCC(Federal Communications Commissions) certified for unlicensed operation. RPi900 provides the ‘glue’ to combine these two devices (DNT900 and RPi) into a ready-to-use package for remote telemetry and automation projects. A power source (like battery), antenna and enclosure are the only extras needed to get it up and running.

RPi900 can remotely power down the Raspberry Pi for further power savings.(To put device on standby mode) RPi900 connects the DNT900 radio to your Raspberry Pi using the ttyAMA0 serial port. By default, the DNT900 is initially configured in transparent mode. In this mode, any characters it receives on the serial lines are transmitted via the radio link to the remote radio. The radio essentially operates as a 'serial cable replacement'.We use the 'stty' command to configure the serial port prior to use in transparent mode. The data rate must be set, which here will be initially set to 9600 b/s (the DNT900 default).

The total data generated by each sensor will be approximately as follows (usable sensor reading required at each instant):

S.No	Sensors	Position	Usage
1.	GPS	3*4 bytes	12 bytes
2.	Thermal sensor	64*4 bytes	256 bytes
3.	IMU	9*4 bytes	36bytes
4.	Radar	just used send warning message so can be assumed 20 bytes	20bytes
5.	LiDAR	1000*6*4 bytes	24000 bytes
		Total	24324 bytes

Which sums to total 190.03125 kb, so our entire data transmission can be completed in almost half of a second. (Here we are doing serial communication so the data is sent bit by bit.) Once the configuration is complete, we can simply send and receive data in the following way:

```
$echo "hello, world" > /dev/ttyAMA0 # send a message
$cat /dev/ttyAMA0   # listen for a response
```

In place of string "hello, world" we can input our data from various sensors like LiDAR, IMU, Thermal sensor, Radar and GPS.
Then this string can be converted to required form using a suitable python code at the receiver's end. For example if we want to send GPS co-ordinates to the controller, then we can send it as:

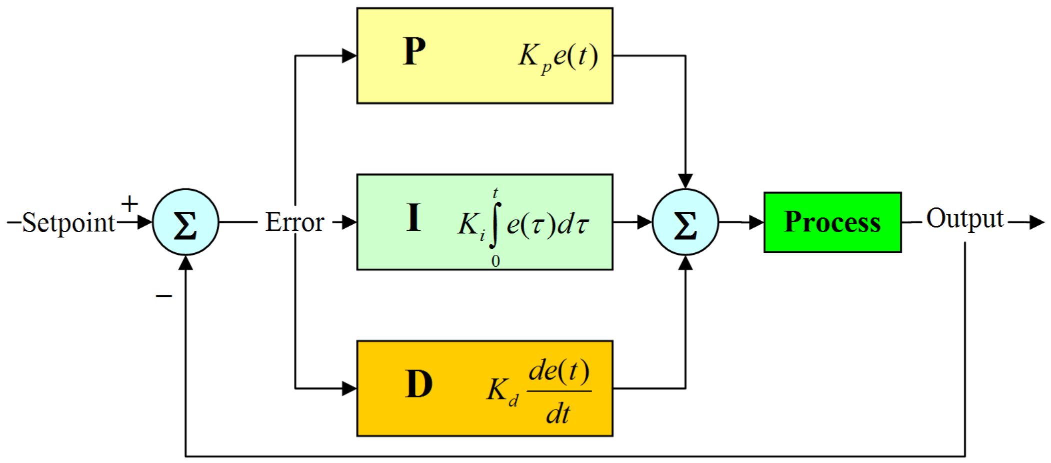
```
$echo "23.4567, 34.4567" > /dev/ttyAMA0 # send a message
$cat /dev/ttyAMA0    # listen for a response
```

Then at the receiver end this data can be accordingly analysed as mentioned in the Remote Controller section . This received data in the form of strings can be converted to an array or can also stored in the form form of a csv file, like in the case of LiDAR.

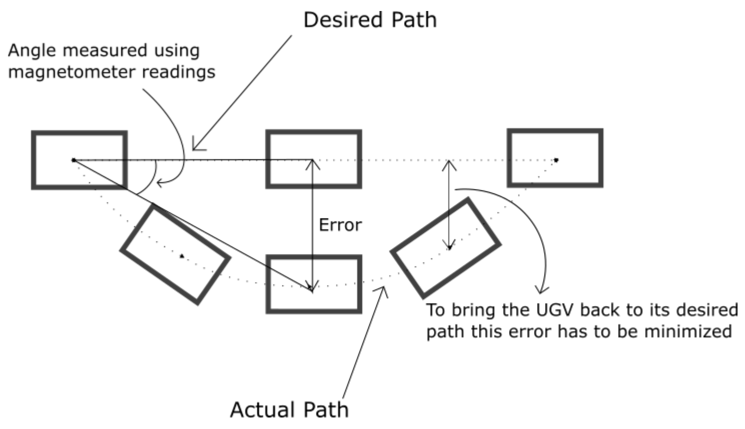
The data from the sensor can be stored in shared memory, with semaphore locks to synchronize read and writes, Whenever a the signal is sent the memory locations are locked to be read by the sender program. thus no data is written there. In addition time stamps can be checked for ensuring that the data being send is new. **MAINTAINING PATH OF THE UGV:**

If the UGV is deviated from its desired path due to some undesired factors like: Skidding due to less friction or Presence of small obstruction like accumulated snow in its path etc.

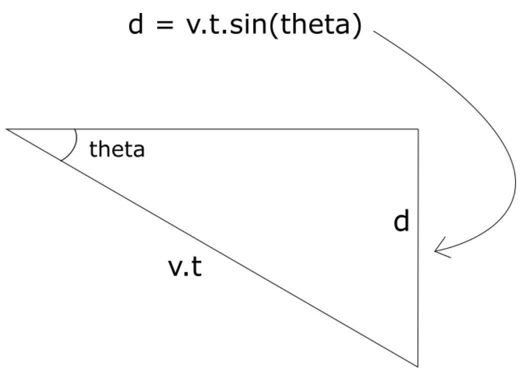
Then to bring the UGV back to its path we intend to use a PID controller.



In our case, the error will be the displacement from the actual path so we will have to minimize this **error** to bring the UGV back to its desired path.


Desired Path
Angle measured using magnetometer readings
Error
To bring the UGV back to its desired path this error has to be minimized
Actual Path

A PID controller has set point (SP) that in our case is 0 displacement from the desired path. The controller output (CO) adjusts the position of rudder to control the direction of motion. In addition, the displacement of the UGV from its desired path at the current state is the process variable (PV) which provides the controller it's-much needed feedback. The output is transmitted in the form instruction to the servo attached to the rudder to rotate through a particular angle. The displacement from the desired path is calculated from the angular displacement determined by the magnetometer readings as follows:



When everything is up and running, the PID controller compares the process variable to its set point and calculates the difference between the two signals, also called the Error (E). Then, based on the Error and the PID controller's tuning constants, the controller calculates an appropriate controller output that rotates the rudder to move the hovercraft in the desired direction. PID controllers have three control modes:

- Proportional Control

- Integral Control
- Derivative Control

Each of the three modes reacts differently to the error. The amount of response produced by each control mode is adjustable by changing the controller's tuning settings. The whole process of maintaining the path by the UGV using a PID has algorithm similar to a Segway based on PID controller for balancing. In that case, the set point is 0 inclination from the vertical axis, controller output is the motor speed and polarity and the feedback is the inclination angle at the current state.
Following is a video of PID controller used on a Segway to move it in a straight line. **(Video link)**

1) ***Remote Controller :*** Once the user gets the required data, the user can control the hovercraft by sending instructions such as moving right or left, increase or decrease speed, activating or deactivating traction system etc.

The instructions will be sent using another RPi900 module attached to the raspberry pi at the user end.

As we are using serial communication so we will send all the instructions at the same time using ";" as delimiter between the instructions and "=" as delimiter to denote the end of each instruction, at the Hovercraft a program will be running on the raspberry pi which will continuously receive these inputs and separate these inputs to send them to respective relays or switches.

All the motors on the Hovercraft will be controlled using 3 arduino boards connected to the raspberry pi.
1st Arduino – controls the traction system
2nd Arduino – controls the rudder
3rd Arduino – controls the Flaps

For the Traction system a Arduino will be used to control the actuators attached to the Hydraulic system. The Arduino will be connected to the raspberry pi on-board which on receiving signal from the user will execute the code on the Arduino required to run the actuators. For controlling the rudders, we are using a servo motor which will have 12 fixed positions to control the direction of motion of hovercraft. For controlling the speed of the hovercraft, we can give analog input to the motor using a motor driver (compatible with the motor) or else we can also use MOSFET and transistor along with a relay to provide variable voltage input.

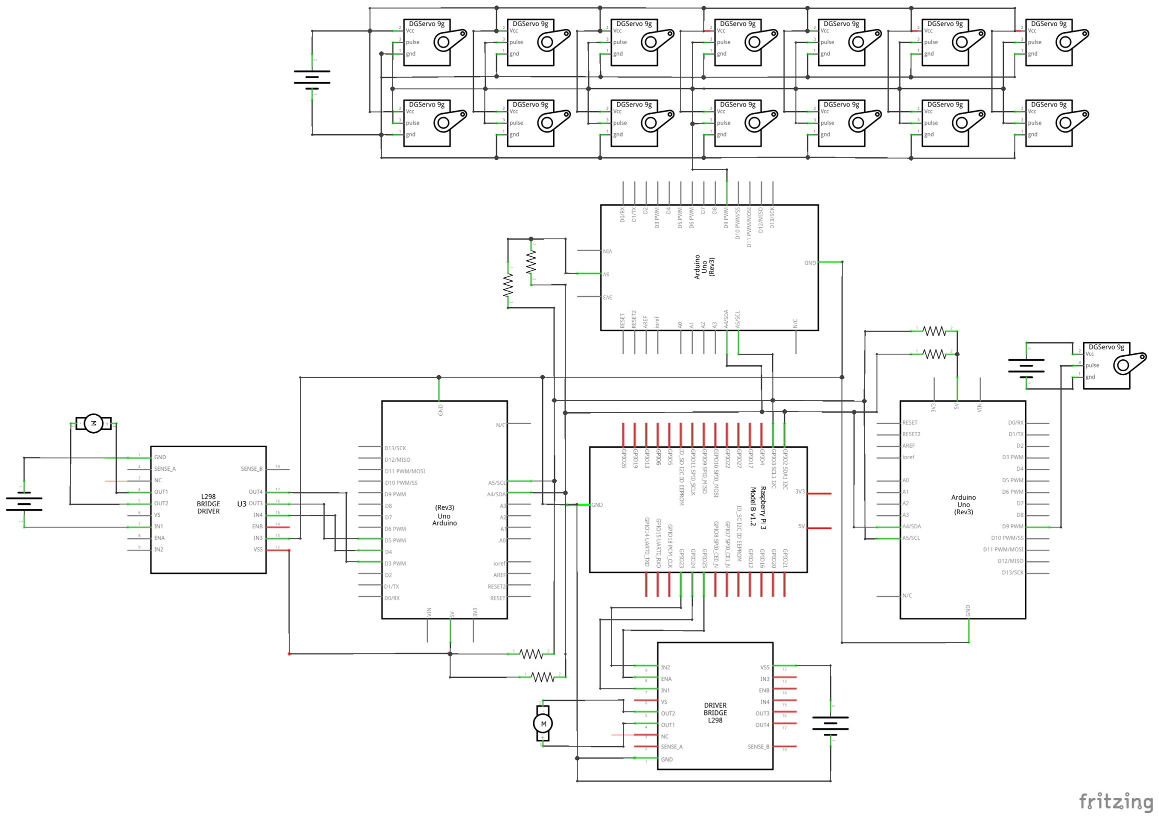
This analog input can be given using a potentiometer in the form of throttle handle at the controller.

The servos can directly be controlled using arduino using their PWM pin. For flaps we are using 14 servos but each servo has to move in the same manner so we can give same input to all the servos.

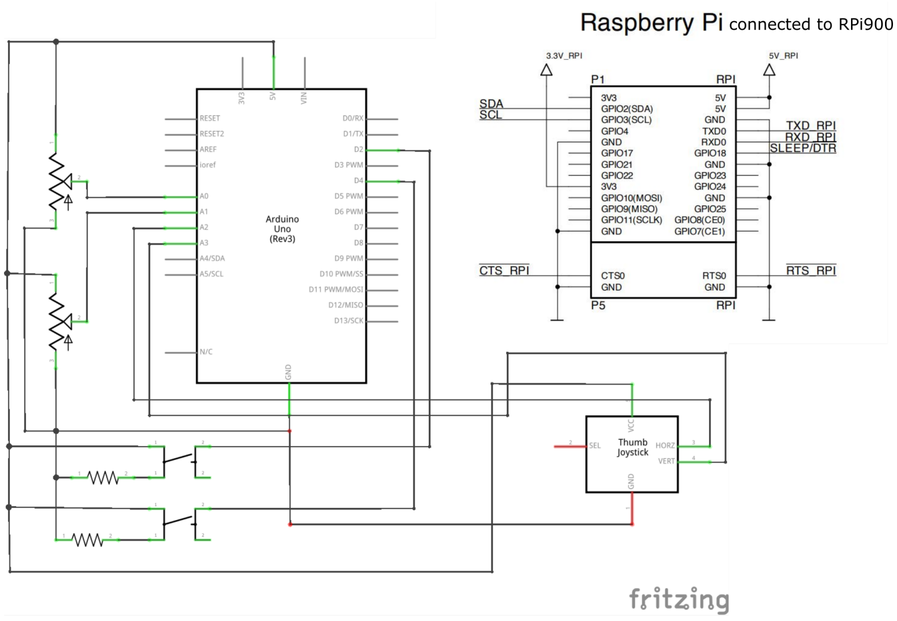
The remote controller with user will consist of following:

- A joystick to give signals to change the direction of UGV.
- Two buttons for retracting and extracting the actuators connected with traction system.
- A potentiometer to control the speed of motor attached with the thrust fan.
- A potentiometer to control the opening and closing of flaps.

All these devices will be connected to a arduino which will take the readings. These readings will then be retrived by raspberry pi using tty/ACM0 port and then transmitted to the UGV in the form of string of characters through serial communication.



Schematic of motor connection on the UGV



Schematic of the controller

Apart from the controlling schematics mentioned above, the controller will be able to look at different graphs showing the data corresponding to the values received from the UGV, The data has to be plotted continuously, to ensure smooth flows , each graph plotting function will be created as a thread, which will wait for user input in the I/O queue before getting executed.

R-Pi has a quad core processor, using multi threading can help us to keep the interface more responsive.

L. *Automated Manoeuvring*

[5] **OverView :** When the vehicle is in motion, there are 3 controlling mechanism that are present
1.The Master controller (Human)
3.The Second Algorithm (Machine Learning System)

When the ACV is on a rescue or supply mission after an avalanche, it will face many issues like maintaining its course in such regions , loss of Visual Signals, loss of Line of sight communication. We intend to handle such scenarios using like maintaining course in such regions by using a PID controller and to decide path to move on using Q-learning.

The motion of vehicles like the hovercraft is very much sensitive to its immediate environment, in the testing done in a simulated environment we found out that to maintain the path of the vehicle extreme amount of experience is required. Building an automated system to take up the task of exploration thus required something more than what the general automated systems like automated cars require, as they have to move in comparatively well-defined terrain roads. With proper marking on the road, hence their task becomes easier.

The PID Controller as discussed in the previous sections can be used to maintain the path of the UGV, in case of obstacle which causes a deviation in the path taken by the UGV the PID controller will come into action to rectify the error.

The Q-learning will be used to decide the direction of manuevering, at any instant of time the system will get a snap shot of the state from the sensors, which will be used to decide the next direction of manouvering. Each state represents a GPS coordinate(latitude,longitude) with a diameter of 3m.

The Q-Learning: Handling a hovercraft is a very daunting task for even a licensed driver of hovercraft , as a lot of its controls depends on the current environmental situations. As hovercrafts hover over the snow, the real area of contact is very small this leads to a situation where every turn can lead to a spinning situation. We try to keep the spinning low by makig sure that the centre of mass is near the propellers, but spinning is usually inevitable. And handling such spins requires expertise. (Design such an algorithm can be a mamoth task).

For such conditions we intend to design machine learnning models which can automaticlly prevent such spinning effect by carefully controlling the fan spped and rudder Each of these mechanism are described in details in the comming sections.

*1) **The Master controller**:* The master controller (Human) will have control of the fan speed ,rudder and air flow control flaps. The sensors will give information about air pressure, velocity and GPS location, LiDAR ,RADAR and Thermal in real time to aid the controllerto take decisions.

*2) **Manoeuvring**:* The terrain in which we expect the proposed hover craft to move are extremely uneven and uncertain. Traditional approaches for this task have proven to be less effective.

We propose the use of a Q learning system to achieve the goal that we have here. Our goal here is to make the system robust enough for the hovercraft to move from point A to point B in a such environment.

The UGV will try to reach the GPS location specified by the controller or go on an exploration mission using this algorithm, The information from the other sensors like optical, LIDAR and RADAR will be used to prevent collisions.

Motivation
We first considered how a human agent might think while going on an exploration mission, one of the basic thing a person does before taking steps is to look at the surroundings, while doing so it calculates the reward it is going to get on interacting with an environment in a particular way.

For example lets take an example of a situation where the human subject is required to move from one point to another. While moving it looks at its immediate surroundings and evaluates the direction

[5]DRUSE-TH02-1339

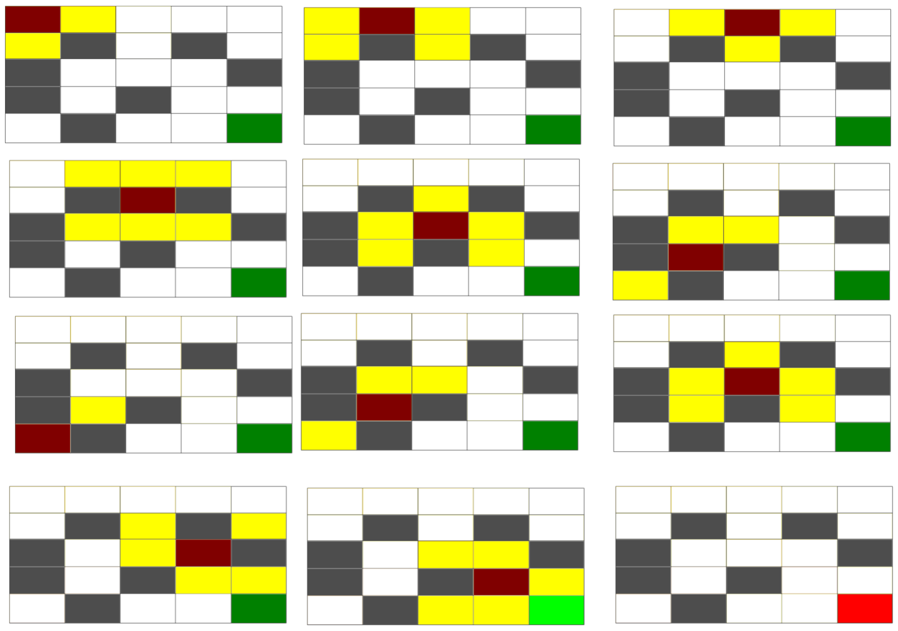
of motion that will yield maximum reward, the human also remembers the previous moves it took and stores rewards related to them with it. Once the reward calculation is done, the human agent takes the best possible action.

Initially when the human agent decides to start its motion, it has little information about the surroundings, so it has to take random actions, it knows that going back is not the optimum option initially as it has to move forward.

Thus, it takes a random action and moves in a random direction, once it reaches the 2nd position, it again analyses the surroundings and takes next action based on where the reward probability is maximum.

Reinforcement Learning in general is based on the idea of the reward hypothesis. All goals can be described by the maximization of the expected cumulative reward. In the diagram below, we can look at the manoeuvring of the human in the grid environment where grey represents obstacles, green represents the goal and brown is the human agent. The area in yellow is what the agent can analyse.

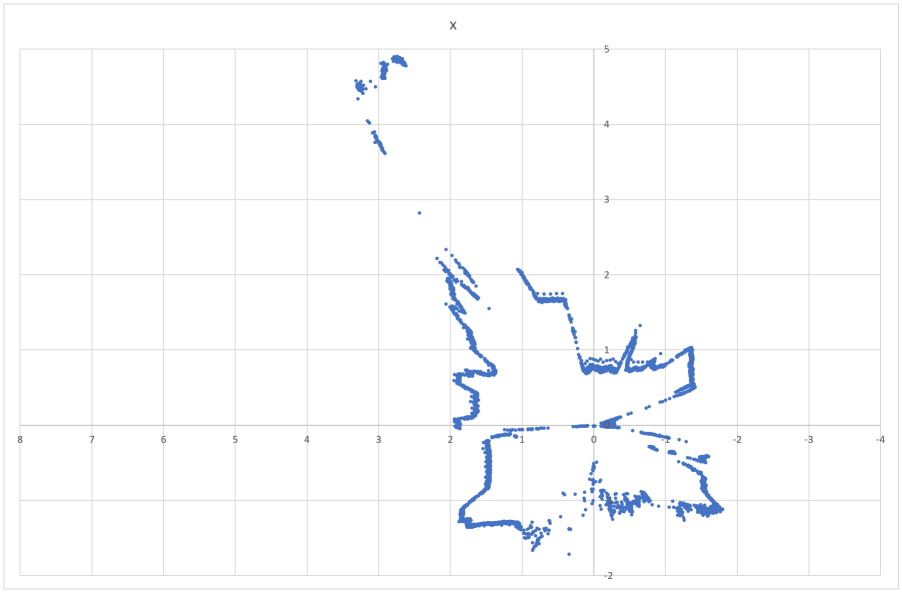
The Machine learning algorithm we intend to use, tries to imitate this behaviour of a human agent to find its path.

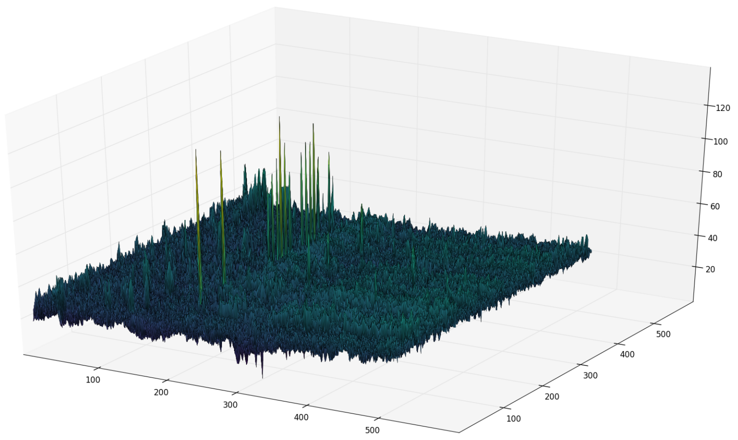


Path taken by a Human Subject

Pre-Requisites

The sensor that is going to be used for this purpose is the LiDAR sensor, The sensor proposed is capable of giving outputs in form given in the image below using which we can generate a matrix containing the obstacles, path, goal and respectively allocate reward to each of the block, These can be used for cumulative reward calculation.





Simulated Environment

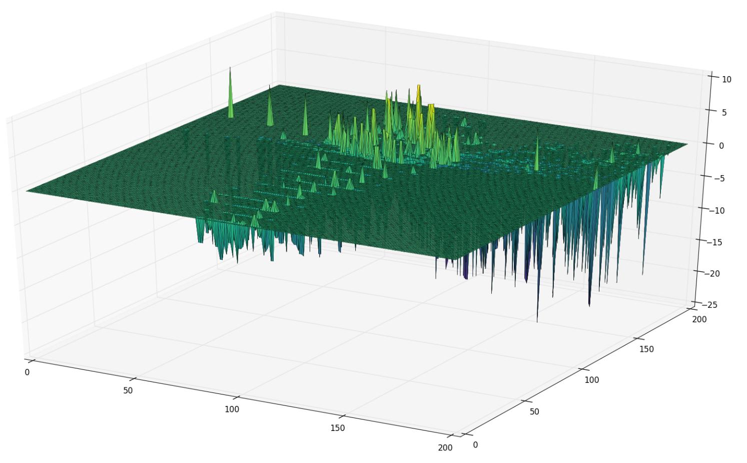
Image from the LiDAR Sensor

Simulating the conditions

In order to simulate the conditions of a natural environment the height-maps of different regions were taken, using their grey scale images. a matrix was formed such that value correspinding to that region contained the elevation of that point.(the environment generated was for testing purpose the maximum height being 5 meters.)

A lidar was simulated to act like a physical lidar, the sensor being simulated here is the velodyne sensor, The sensor beams out 32 beams ranging over an angle of -30 degree to 10 degree in the XZ place , distributed uniformly in the range specified before. The sensor rotates in the XY plane and can rotate at speed of 10 Hz, For the purpose of simulation we take the reading at each instance once. 32 such readings are calculated 360 time at 360 different angle uniformly ranging from 0 degree to 360 degree i.e. at each degree 32 readings are calculated. The matrix thus formed is of size 360 X 32 an example matrix for the readings is as such. It is important to note here that the ith row and jth column represents the distance of the obstacle from the sensor in the XY plane(sqrt(square(x))+square(y)))

The algorithm used for making the LiDAR is as described in Appendix:



Simulated readings from LiDAR

We first calculate the the y coordinate based on the position of the sensor and the slope of the line corresponding to the angle take in the XY plane (0-360 degree),

once the slop is decided, the corresponding value of z is obtained from the matrix of heightmaps(the height at a distance x from the y axis and at a diatance y from the x axis).

Now we select 32 possible angles and check if the given point (X,Z) lie on any of this line with a possible error of 0.5 units. starting from -30 degree .

Running this loop for multiple times we finally abtain the matrix where each row represents an angle between 0-360 degrees and each column represents one of the 32 beams. the vaule of ith row and jth column contains the distance of the obstacle from the sensor an example output for a 32 beams for a given angle on a plane surface is described as shown in the image

Thus obtained 360 X 32 matrix can now be converted into graph of one such angle(XZ plane) as shown below.

The obtained graph can then be utilized to create a matrix with rewards and penalties based on height of the obstacles similar to the examples shown above. The LiDAR wil be place 60 cm above base of the Hovercraft when the skirt is completely inflated, it will be present at a distance of 30 cm from the front edge of the hovercraft.Thus Z will always remain at a minimum of 60 cm from the ground when in action.

How to decide which path is favourable ?

Now once the environment is created, Now how to decide which path is favourable. For human agent the brain does the complicated part of deciding the path based on expected rewards and prior experiences,we will get into doing these 2 parts mathematically in a later sections:

Deciding the path:

Reinforcement Learning is based on the idea of the reward hypothesis. All goals can be described by the maximization of the expected cumulative reward. The reward that are far from us are less probable to be achieved compared to the rewards near us, Thus we take care of such cases by using a constant value gamma (γ) this constant will help us weight the rewards according to their distance from the agent. In our case the cumulative reward will be as given below:

$$cumulativeReward = \gamma R_1 + \gamma^2 R_2 + \gamma^3 R_3 + \gamma^4 R_4 + ...$$

Here R is the reward related to each position. In our case the slopes will decide the rewards a steep fall is a penalty , a tall obstacle is a penatly too.

Before looking at the different strategies to solve our Reinforcement Learning problems, we must take care of one more very important topic the exploration/exploitation trade-off.

Exploration is finding more information about the environment whereas Exploitation is exploiting known information to maximize the reward. The goal of our Reinforcement learning model agent is to maximize the expected cumulative reward. However, we can fall into a common trap. Like what should the model do for its first turn, when it should explore when it should not explore and exploit instead.

For our purpose the goal is to optimize a value function V(s), The value function is a function that tells us the maximum expected future reward the agent will get at each state. The value of each state is the total amount of the reward an agent can expect to accumulate over the future, starting at that state.

$$V(s) = E[\gamma R_1 + \gamma^2 R_2 + \gamma^3 R_3 + \gamma^4 R_4 + ...|State = s]$$

The agent will use this value function to give value to each state. A detailed calculation is given in a later section where we will create a table to calculate the maximum expected future reward, for each action at each state.The model of hovercraft here will do a similar action at each instance of time (with a frequency of 1 second the complete break down of the time taken is described in a later section), using which the agent will know what's the best action to take at each state. We can transform this grid into a table.

The table is called a Q-table ("Q" for "quality" of the action). The column her will be the actions. The rows will be the states(Latitude and longitude in pur case). The value of each cell will be the maximum expected future reward for that given state and action.

Each Q-table score will be the maximum expected future reward that it will get if I take that action at that state with the best policy given. The policy here is to take the action with the maximum possible reward, which depends on the value of V(s) of each state and the reward. Q-table is like a "cheat sheet." which helps us to take the best action.
Now the value for each state action pair in the Q-table can be calculated using the Q learning algorithm.Based on Bellaman's Equation explained in a later section.

The q function for a given state and actionn pai is gien by:

$$Q(s_t, a_t) = E[\gamma R_1 + \gamma^2 R_2 + \gamma^3 R_3 + \gamma^4 R_4 + ...|s_t, a_t]$$

In the earlier part we mentiond the exploration exploitation trade-off, to take care of it we introduce anothe constant say epsilon, in the starting we will have to do more exploration but one we get more amiliar with the envirronment the exploration (Random moves) needs to be decreased and we haveto take defined actions based on the q table.

To do so we have to use a decay rate for the epsilon constant , after each instance the epsiplon will decrease and and hence the possibility of taking radom ction.

How are we going to use this in our algrithm, at each instance random value will be generated if that value is smaller than the epsilon then a random action will be takn else the actionwill be taken accordind to the q table.

Becasuse of the decay function slowly the exploration will decrease and exploitation will increase.

Getting back to the first Q table,we update the values of the q table using the followingequation called the bellaman$'$s equation as shown below :

h i

$$UpdatedQ(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma max(Q'(s',a')) - Q(s,a)]$$

The psuedo code for the whole process described above is given in Appendix 9, Once we obtain the data from the lidar and make the required changes.

*M. **If qualified for the second level of screening what Idea/Concept/Solution we are proposing?***

We intend to go more deeper in the area of machine learning to stabilize the model and explain further on the aerodynamic and steering of the mode.

*N. **If qualified for the third level of screening what Idea/Concept/Solution we are proposing?***

We intend to present a complete solution for the problem with complete details of specifications and materials to be used. To actually build the UGV and The trained machine learning model and optimization funnels to be utilized for assistive driving. [6]

*O. **What is our plan for your concept to mature into a realizable prototype?***

The concept can be matured in a realizable model , a prototype will be presented by us which will be a scaled down version of the model we are proposing here.

[6]DRUSE-TH02-1339

Appendix

Appendix 1

```
#!/usr/bin/python
# Copyright (c) 2017 Adafruit Industries
# Author: Dean Miller
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the ”Software”), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED ”AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.

# Can enable debug output by uncommenting:
#import logging
#logging.basicConfig(level=logging.DEBUG)

from Adafruit_AMG88xx import Adafruit_AMG88xx
from time import sleep

#import Adafruit_AMG88xx.Adafruit_AMG88xx as AMG88

# Default constructor will pick a default I2C bus.
#
# For the Raspberry Pi this means you should hook up to the only exposed I2C bu
# from the main GPIO header and the library will figure out the bus number base
# on the Pi's revision.
#
# For the Beaglebone Black the library will assume bus 1 by default, which is
# exposed with SCL = P9_19 and SDA = P9_20.
sensor = Adafruit_AMG88xx()

# Optionally you can override the bus number:
#sensor = AMG88.Adafruit_AMG88xx(busnum=2)

#wait for it to boot
sleep(.1)
```

```
while(1):
        print(sensor.readPixels())
        sleep(1)
```

Appendix 2

```
from Adafruit_AMG88xx import Adafruit_AMG88xx
import pygame
import os
import math
import time

import numpy as np
from scipy.interpolate import griddata

from colour import Color

#low range of the sensor (this will be blue on the screen)
MINTEMP = 26

#high range of the sensor (this will be red on the screen)
MAXTEMP = 32

#how many color values we can have
COLORDEPTH = 1024

os.putenv('SDL_FBDEV', '/dev/fb1')
pygame.init()

#initialize the sensor
sensor = Adafruit_AMG88xx()

points = [(math.floor(ix / 8), (ix % 8)) for ix in range(0, 64)]
grid_x, grid_y = np.mgrid[0:7:32j, 0:7:32j]

#sensor is an 8x8 grid so lets do a square
height = 240
width = 240

#the list of colors we can choose from
blue = Color("indigo")
colors = list(blue.range_to(Color("red"), COLORDEPTH))

#create the array of colors
colors = [(int(c.red * 255), int(c.green * 255), int(c.blue * 255)) for c in c

displayPixelWidth = width / 30
displayPixelHeight = height / 30

lcd = pygame.display.set_mode((width, height))

lcd.fill((255,0,0))

pygame.display.update()
pygame.mouse.set_visible(False)

lcd.fill((0,0,0))
pygame.display.update()

#some utility functions
def constrain(val, min_val, max_val):
    return min(max_val, max(min_val, val))

def map(x, in_min, in_max, out_min, out_max):
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

#let the sensor initialize
time.sleep(.1)

while(1):

        #read the pixels
        pixels = sensor.readPixels()
        pixels = [map(p, MINTEMP, MAXTEMP, 0, COLORDEPTH - 1) for p in pixels]

        #perdorm interpolation
        bicubic = griddata(points, pixels, (grid_x, grid_y), method='cubic')

        #draw everything
        for ix, row in enumerate(bicubic):
                for jx, pixel in enumerate(row):
                        pygame.draw.rect(lcd, colors[constrain(int(pixel), 0, C

        pygame.display.update()
```

Appendix 3

```
# Distributed with a free-will license.
# Use it any way you want, profit or free, provided it fits in the licenses of
# LSM9DS0
# This code is designed to work with the LSM9DS0_I2CS I2C Mini Module available
# https://www.controleverything.com/content/Accelorometer?sku=LSM9DS0_I2CS#tabs

import smbus
import time

# Get I2C bus
bus = smbus.SMBus(1)

# LSM9DS0 Gyro address, 0x6A(106)
# Select control register1, 0x20(32)
#               0x0F(15)        Data rate = 95Hz, Power ON
#                                       X, Y, Z-Axis enabled
bus.write_byte_data(0x6A, 0x20, 0x0F)
# LSM9DS0 address, 0x6A(106)
# Select control register4, 0x23(35)
#               0x30(48)        DPS = 2000, Continuous update
bus.write_byte_data(0x6A, 0x23, 0x30)

time.sleep(0.5)

# LSM9DS0 Gyro address, 0x6A(106)
# Read data back from 0x28(40), 2 bytes
# X-Axis Gyro LSB, X-Axis Gyro MSB
data0 = bus.read_byte_data(0x6A, 0x28)
data1 = bus.read_byte_data(0x6A, 0x29)

# Convert the data
xGyro = data1 * 256 + data0
if xGyro > 32767 :
        xGyro -= 65536

# LSM9DS0 Gyro address, 0x6A(106)
# Read data back from 0x2A(42), 2 bytes
# Y-Axis Gyro LSB, Y-Axis Gyro MSB
data0 = bus.read_byte_data(0x6A, 0x2A)
data1 = bus.read_byte_data(0x6A, 0x2B)

# Convert the data
yGyro = data1 * 256 + data0
if yGyro > 32767 :
        yGyro -= 65536

# LSM9DS0 Gyro address, 0x6A(106)
# Read data back from 0x2C(44), 2 bytes
# Z-Axis Gyro LSB, Z-Axis Gyro MSB
data0 = bus.read_byte_data(0x6A, 0x2C)
data1 = bus.read_byte_data(0x6A, 0x2D)

# Convert the data
zGyro = data1 * 256 + data0
if zGyro > 32767 :
        zGyro -= 65536

# LSM9DS0 Accl and Mag address, 0x1E(30)
# Select control register1, 0x20(32)
#               0x67(103)       Acceleration data rate = 100Hz, Power ON
#                                               X, Y, Z-Axis enabled
bus.write_byte_data(0x1E, 0x20, 0x67)
# LSM9DS0 Accl and Mag address, 0x1E(30)
# Select control register2, 0x21(33)
#               0x20(32)        Full scale = +/-16g
bus.write_byte_data(0x1E, 0x21, 0x20)
# LSM9DS0 Accl and Mag address, 0x1E(30)
# Select control register5, 0x24(36)
#               0x70(112)       Magnetic high resolution, Output data rate = 5
bus.write_byte_data(0x1E, 0x24, 0x70)
# LSM9DS0 Accl and Mag address, 0x1E(30)
# Select control register6, 0x25(37)
#               0x60(96)        Magnetic full scale selection = +/-12 gauss
bus.write_byte_data(0x1E, 0x25, 0x60)
# LSM9DS0 Accl and Mag address, 0x1E(30)
# Select control register7, 0x26(38)
#               0x00(00)        Normal mode, Magnetic continuous conversion mo
bus.write_byte_data(0x1E, 0x26, 0x00)

time.sleep(0.5)

# LSM9DS0 Accl and Mag address, 0x1E(30)
# Read data back from 0x28(40), 2 bytes
# X-Axis Accl LSB, X-Axis Accl MSB
data0 = bus.read_byte_data(0x1E, 0x28)
data1 = bus.read_byte_data(0x1E, 0x29)

# Convert the data
xAccl = data1 * 256 + data0
if xAccl > 32767 :
        xAccl -= 65536

# LSM9DS0 Accl and Mag address, 0x1E(30)
# Read data back from 0x2A(42), 2 bytes
# Y-Axis Accl LSB, Y-Axis Accl MSB
data0 = bus.read_byte_data(0x1E, 0x2A)
data1 = bus.read_byte_data(0x1E, 0x2B)
```

```
# Convert the data
yAccl = data1 * 256 + data0
if yAccl > 32767 :
        yAccl -= 65536

# LSM9DS0 Accl and Mag address, 0x1E(30)
# Read data back from 0x2C(44), 2 bytes
# Z-Axis Accl LSB, Z-Axis Accl MSB
data0 = bus.read_byte_data(0x1E, 0x2C)
data1 = bus.read_byte_data(0x1E, 0x2D)

# Convert the data
zAccl = data1 * 256 + data0
if zAccl > 32767 :
        zAccl -= 65536

# LSM9DS0 Accl and Mag address, 0x1E(30)
# Read data back from 0x08(08), 2 bytes
# X-Axis Mag LSB, X-Axis Mag MSB
data0 = bus.read_byte_data(0x1E, 0x08)
data1 = bus.read_byte_data(0x1E, 0x09)

# Convert the data
xMag = data1 * 256 + data0
if xMag > 32767 :
        xMag -= 65536

# LSM9DS0 Accl and Mag address, 0x1E(30)
# Read data back from 0x0A(10), 2 bytes
# Y-Axis Mag LSB, Y-Axis Mag MSB
data0 = bus.read_byte_data(0x1E, 0x0A)
data1 = bus.read_byte_data(0x1E, 0x0B)

# Convert the data
yMag = data1 * 256 + data0
if yMag > 32767 :
        yMag -= 65536

# LSM9DS0 Accl and Mag address, 0x1E(30)
# Read data back from 0x0C(12), 2 bytes
# Z-Axis Mag LSB, Z-Axis Mag MSB
data0 = bus.read_byte_data(0x1E, 0x0C)
data1 = bus.read_byte_data(0x1E, 0x0D)

# Convert the data
zMag = data1 * 256 + data0
if zMag > 32767 :
        zMag -= 65536
```

```
# Output data to screen
print "X-Axis of Rotation : %d" %xGyro
print "Y-Axis of Rotation : %d" %yGyro
print "Z-Axis of Rotation : %d" %zGyro
print "Acceleration in X-Axis : %d" %xAccl
print "Acceleration in Y-Axis : %d" %yAccl
print "Acceleration in Z-Axis : %d" %zAccl
print "Magnetic field in X-Axis : %d" %xMag
print "Magnetic field in Y-Axis : %d" %yMag
print "Magnetic field in Z-Axis : %d" %zMag

#Output data to file
```

Appendix 4

```
def getrmc():

    serial = wp.serialOpen("/dev/ttyAMA0",57600) # open serial port

    wp.serialFlush(serial)
    print(serial)
    while True: # repeat until we get a RMC NMEA string
        gpsstring = ""
        while True: # repeat until we have a complete string
            if (wp.serialDataAvail(serial) > 0):
        letter = wp.serialGetchar(serial)
                if letter == 10:
                    break
            else:
                    gpsstring += str(chr(letter))
        if (gpsstring[3:6]=="RMC"):
            break
    wp.serialClose(serial)
    return(gpsstring)
```

Appendix 5

```
def cmd_read_GPS():
    reading=getrmc()
    # determine the position of the value separating commas
    commas = [0,1,2,3,4,5,6,7,8,9,10,11]
    comnum = 0
    for i in range (0,len(reading)):
        if reading[i] == ",":
            commas[comnum] = i # save the position of the comma
            comnum += 1
    # Extract latitude
    if (reading[commas[3] + 1:commas[4]]) == "N":
        sign = 1
    else:
        sign = -1

    degrees = float(reading[commas[2]+1:commas[2]+3])
    minutes = float(reading[commas[2] + 3:commas[3]])/60
    latitude = sign*(degrees + minutes)
    # Extract Longitude
    if (reading[commas[5] + 1:commas[6]]) == "E":
        sign = 1
    else:
        sign = -1

    degrees = float(reading[commas[4]+1:commas[4]+3])
    minutes = float(reading[commas[4] + 3:commas[5]])/60
    longitude = sign*(degrees + minutes)


    return(str(latitude),",",str(longitude))
```

Appendix 6

```
#!/usr/bin/env python

from __future__ import division
import serial
import math
import os
import struct

with serial.Serial("/dev/ttyUSB0",
                    baudrate = 115200,
                    parity=serial.PARITY_NONE,
                    bytesize = serial.EIGHTBITS,
                    stopbits = serial.STOPBITS_ONE,
                    xonxoff = False,
                    rtscts = False,
                    dsrdtr = False) as sweep:

    print "Scanse Sweep open"
    sweep.write("ID\n")
    print "Query device information"
    resp = sweep.readline()
    print "Response: " + resp

    print "Starting scanning...",
    sweep.write("DS\n")
    resp = sweep.readline()
    assert (len(resp) == 6), "Bad data"

    status = resp[2:4]
    if  status == "00":
        print "OK"
    else:
        print "Failed %s" % status

        #-----------------------------------------------------------------------
        # Missing here is stopping the scanning - it will still be running nex
        # time code is initiated and it all gets very messy / confusong separat
        # binary data from ASCII command / response.  Really need to do a subs
        # the finally: branch below.
        #-----------------------------------------------------------------------
        os.exit()

    log = open("sweep.csv", "wb")
    log.write("angle, distance, x, y\n")

    format = '=' + 'B' * 7

    try:
```

```
    while True:
        line = sweep.read(7)
        assert (len(line) == 7), "Bad data read: %d" % len(line)
        data = struct.unpack(format, line)
        assert (len(data) == 7), "Bad data type conversion: %d" % len(data

        azimuth_lo = data[1]
        azimuth_hi = data[2]
        angle_int = (azimuth_hi << 8) + azimuth_lo
        degrees = (angle_int >> 4) + (angle_int & 15) / 16

        distance_lo = data[3]
        distance_hi = data[4]
        distance = ((distance_hi << 8) + distance_lo) / 100

        x = distance * math.cos(degrees * math.pi / 180)
        y = distance * math.sin(degrees * math.pi / 180)

        log.write("%f, %f, %f, %f\n" % (degrees, distance, x, y))

#----------------------------------------------------------------------------
# Catch Ctrl-C
#----------------------------------------------------------------------------
except KeyboardInterrupt as e:
    pass

#----------------------------------------------------------------------------
# Catch incorrect assumption bugs
#----------------------------------------------------------------------------
except AssertionError as e:
    print e

#----------------------------------------------------------------------------
# Cleanup regardless otherwise the next run picks up data from this
#----------------------------------------------------------------------------
finally:
    print "Stop scanning"
    sweep.write("DX\n")
    resp = sweep.read()
    print "Response: %s" % resp
    log.close()
```

Appendix 7

```
#Libraries
import RPi.GPIO as GPIO
import time

#GPIO Mode (BOARD / BCM)
GPIO.setmode(GPIO.BCM)

#set GPIO Pins
GPIO_TRIGGER = 18
GPIO_ECHO = 24

#set GPIO direction (IN / OUT)
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)

def distance():
    # set Trigger to HIGH
    GPIO.output(GPIO_TRIGGER, True)

    # set Trigger after 0.01ms to LOW
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)

    StartTime = time.time()
    StopTime = time.time()

    # save StartTime
    while GPIO.input(GPIO_ECHO) == 0:
        StartTime = time.time()

    # save time of arrival
    while GPIO.input(GPIO_ECHO) == 1:
        StopTime = time.time()

    # time difference between start and arrival
    TimeElapsed = StopTime - StartTime
    # multiply with the sonic speed (34300 cm/s)
    # and divide by 2, because there and back
    distance = (TimeElapsed * 34300) / 2

    return distance

if __name__ == '__main__':
    try:
        while True:
            dist = distance()
            print ("Measured Distance = %.1f cm" % dist)
            time.sleep(1)
    # Reset by pressing CTRL + C
except KeyboardInterrupt:
    print("Measurement stopped by User")
    GPIO.cleanup()
```

Appendix 8

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import math
from mpl_toolkits.mplot3d import Axes3D

# All units are in meter and radians respecitively

Z_SAMPLES = 32
Y_SAMPLES = 360
HEIGHT_SENSOR = 1.12
MAX_ANGLE =  0.18622663
MIN_ANGLE = -0.53529248
MIN_DISTANCE = 0.05
MAX_DISTANCE = 70

XZ_LEAST_COUNT = (MAX_ANGLE - MIN_ANGLE)/Z_SAMPLES
Y_LEAST_COUNT = 0.0174533



def y_slope(num):
    slope = math.tan( Y_LEAST_COUNT*num)
    return slope

def y_angle(num):
    angle = ( Y_LEAST_COUNT*num)
    return angle


def z_slope(b_num):
    slope = math.tan(MIN_ANGLE + XZ_LEAST_COUNT*b_num)
    return slope

def point_in_line(z,m,x,c):
    a = abs(z - (m*x) - c) / (math.sqrt(1.0+(m**2.0)))

    if(a<0.5):
        return True
    else:
        return False

def one_beam(b_num, x_pos, y_pos, z_pos, y_num, data):
    slope =  z_slope(b_num)

    constant = z_pos

    yslope =  y_slope(y_num)
    if(y_angle(y_num) <= 1.57 or y_angle(y_num) >= 4.71 ):
        for i in range(x_pos,x_pos+MAX_DISTANCE):
            y = int(yslope*(i-x_pos))+y_pos
            if( i<0 or i >= 256 or y>= 256 or y<0):
                continue
            dist = math.sqrt(((i-x_pos)**2)+((y-y_pos)**2))
            print(i,y)
            z_value = data[i][y]
            if(point_in_line(z_value,slope,i-x_pos,constant)):
                return([dist])
        return [0]
    else:
        for i in range(x_pos-MAX_DISTANCE,x_pos):
            y = int(yslope*(i-x_pos))+y_pos
            if(i<0 or i >= 256 or y>= 256 or y<0):
                continue
            dist = math.sqrt(((i-x_pos)**2)+((y-y_pos)**2))
            z_value = data[i][y]
            if(point_in_line(z_value,slope,i-x_pos,constant)):
                return([dist])
        return [0]


def lidar_read(x_pos,y_pos,z_pos,data):
    lidar_val = [[0 for i in range(Z_SAMPLES)] for j in range(Y_SAMPLES)]

    for i in range(Y_SAMPLES):
        for j in range(Z_SAMPLES):
            a = one_beam(j,x_pos,y_pos,z_pos,i,data)
            lidar_val[i][j] = a[0]
    return lidar_val

def main():
    data = np.genfromtxt('test.csv', delimiter=',')
    data = ((data-np.mean(data))/np.std(data))
    data = data*5
    x_pos=25
    y_pos=25
    z_pos=5
    lidar = lidar_read(x_pos,y_pos,z_pos,data)
    mix = [[0 for i in range(Z_SAMPLES)] for j in range(Y_SAMPLES)]

    for i in range (Y_SAMPLES):
        for j in range(Z_SAMPLES):
                mix[i][j] = np.sum(lidar[i][j])/np.size(lidar[0][0])


    plt.plot(mix[0][:],linestyle='--', marker='o')
    plt.xlabel("Angles on the X-Z Planes")
    plt.ylabel("Distance of the obstacle from the Sensor(units)")
    plt.show()

    fmix = [[0 for i in range(200)] for j in range(200)]
    for i in range(360):
        for j in range(32):
            slope = y_slope(i)
            y_ = slope*mix[i][j]+100
            if(y_<0 or int(y_)>=200):
                continue
            fmix[int(y_)][int(mix[i][j])] = mix[i][j]*z_slope(j) +10


main()
data = pd.read_csv('check.csv')

# Transform it to a long format
df=data.unstack().reset_index()
df.columns=["X","Y","Z"]

# And transform the old column name in something numeric
df['X']=pd.Categorical(df['X'])
df['X']=df['X'].cat.codes

# Make the plot
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_trisurf(df['Y'], df['X'], df['Z'], cmap=plt.cm.viridis, linewidth=0.2)
plt.show()
```

Appendix 9

```
import numpy as np
import math
import random

data = np.genfromtxt('test.csv', delimiter=',')
x_pos = 0
y_pos = 0
epsilon = 1
alpha=0.6
max_epsilon = 1
min_epsilon = 0.01
decay_rate = 0.01
gamma = 0.9
q_table = [[0 for i in range(np.shape(data)[0])] for j in range(np.shape(data)

def score( data):
    data = ((data-np.mean(data))/np.std(data))
    return  data

def cal_q(i,j,a,b):

    if(abs(i-a)>2 or abs(a-i) > 2 or abs(b-j) >2 or abs(j-b) >2 or a<0 or b <
        return 0

    try:
        p = cal_q(i,j,a+1,b)
    except:
        p =-1
    q = cal_q(i,j,a-1,b)
    s = cal_q(i,j,a,b-1)
    try:
        r = cal_q(i,j,a,b+1)
    except:
        r = -1

    q_table[a][b] = q_table[a][b] + alpha*((data[a][b] + gamma*np.max([p,q,r,s
    return  q_table[a][b] + alpha*((data[a][b] + gamma*np.max([p,q,r,s]) ) - q


def main():

    score(data)
    position = 0
    x_pos = 0
    y_pos = 0
    epsilon = 1
    for itere in range(100) :
        if (random.random() < epsilon ):
```

```
        epsilon = min_epsilon + (max_epsilon − min_epsilon)∗np.exp(−decay_
        choice = random.randint(0,4)
        print(choice)
        if (choice == 0 or x_pos <256):
            x_pos = x_pos+1

        elif (choice == 1 or x_pos >=0 ):
            x_pos = x_pos−1

        elif (choice == 2 or y_pos <256):
            y_pos = y_pos+1

        elif (choice == 3 or y_pos >=0 ):
            y_pos = y_pos−1
        position = data[x_pos][y_pos]
        i = x_pos
        j = y_pos
        q_table[i][j] =cal_q(i,j,i,j)
        print(x_pos,y_pos)
        continue
    print("Not Random")
    i = x_pos
    j = y_pos
    q_table[i][j] =cal_q(i,j,i,j)
    if(i+1<  np.shape(data)[0] or i+1>0 or j>= 0 or i >= 0 ):

        a =  q_table[i+1][j]
    else:
        a =−1
    if(i−1<np.shape(data)[0] or i−1>0 or j>=0 or i >= 0 ):
        b = q_table[i−1][j]
    else:
        b = −1
    if(j+1<np.shape(data)[1] or j+1>0 or j >= 0 or i >= 0 ):
        c = q_table[i][j+1]
    else:
        c = −1
    if(j−1<np.shape(data)[1] or j−1>0 or j >= 0 or i >= 0):
        d = q_table[i][j−1]
    else:
        d = −1

    print(a,b,c,d)
    choice = np.argmax([a,b,c,d])

    if (choice == 0):
        x_pos = i+1
        y_pos = j
    if (choice == 1):
            x_pos = i-1
            y_pos = j
        if (choice == 2):
            x_pos = i
            y_pos = j+1
        if (choice == 3):
            x_pos = i
            y_pos = j-1
        print(x_pos,y_pos)

main()
```

Appendix 10

```
# library
from PIL import Image
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np
import sys
image = sys.argv[1]
ext = '.'+sys.argv[2]

im = Image.open(image+ext,'r')

width = 256
height = 256
mix = [[0 for i in range(width)] for j in range(height)]
im2 = im.resize((width, height), Image.NEAREST)      # use nearest neighbour
im3 = im.resize((width, height), Image.BILINEAR)     # linear interpolation in
im4 = im.resize((width, height), Image.BICUBIC)      # cubic spline interpolati
im = im.resize((width, height), Image.ANTIALIAS)    # best down-sizing filter

im.save(image+'_check' + ext)
pix = im.load()
print(np.size(pix[0,0]))
for i in range (width):
    for j in range(height):
            mix[i][j] = np.sum(pix[i,j])/np.size(pix[0,0])
mix = mix-np.mean(mix)/np.std(mix)
np.savetxt(image+'.csv',mix,delimiter=',')

data = pd.read_csv('test0.csv')

# Transform it to a long format
df=data.unstack().reset_index()
df.columns=["X","Y","Z"]
```

```
# And transform the old column name in something numeric
df['X']=pd.Categorical(df['X'])
df['X']=df['X'].cat.codes

# Make the plot
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_trisurf(df['Y'], df['X'], df['Z'], cmap=plt.cm.viridis, linewidth=0.2)
plt.show()
```

Appendix 11

```
int xVal; //X values from joystick

int yVal; //Y values from joystick

void setup() {

Serial.begin(9600); //Starts serial at 9600 baud

pinMode(A0, INPUT); //Sets the analog ports used to an input

pinMode(A2, INPUT);

}

void loop() {

xVal = analogRead(A0); //sets the X value

yVal = analogRead(A2); //sets the Y value

Serial.print(” Y is...”);

Serial.print(yVal);} //prints Y values

Serial.print(” X is...”);

Serial.print(xVal);}} //prints X values
```

Appendix 12

```
// Arduino linear actuator demo
// dan@marginallyclever.com 2015-07-28
// http://www.github.com/MarginallyClever/vnh5019_linear_actuator

void setup() {
  Serial.begin(57600);

  // put your setup code here, to run once:
  pinMode(2,OUTPUT);
  pinMode(3,OUTPUT);
  pinMode(4,OUTPUT);

  Serial.println("** Ready **");
}

void loop() {
  Serial.println("Slide Forward");
  setDir(1);
  slide(5);

  Serial.println("Slide Backward");
  setDir(2);
  slide(5);
}

void setDir(int d) {
  switch(d) {
    case 0: // off?
    digitalWrite(2,LOW);
    digitalWrite(4,LOW);
    break;
    case 1: // forward
    digitalWrite(2,HIGH);
    digitalWrite(4,LOW);
    break;
    case 2:  // backward
    digitalWrite(2,LOW);
    digitalWrite(4,HIGH);
    break;
    case 3:  // locked?
    digitalWrite(2,HIGH);
    digitalWrite(4,HIGH);
    break;
  }
}


// gradually speed up, then gradually slow down.
```

```
void slide(int d) {
  for(int v=0;v<256;++v) {
    analogWrite(3,v);
    delay(d);
  }
  for(int v=255;v>=0;--v) {
    analogWrite(3,v);
    delay(d);
  }
```

Appendix 13

```
#include          //Servo library

 Servo servo_test;      //initialize a servo object for the connected servo

 int angle = 0;
 int potentio = A0;      // initialize the A0analog pin for potentiometer


 void setup()
 {
  servo_test.attach(9);   // attach the signal pin of servo to pin9 of arduino
 }

 void loop()
 {
  angle = analogRead(potentio);            // reading the potentiometer value b
  angle = map(angle, 0, 1023, 0, 179);     // scaling the potentiometer value t
  servo_test.write(angle);                   //command to rotate the servo to t
  delay(5);
 }
```