1. INTRODUCTION

1.1. INTRODUCTION TO THE SCOPE OF WORK

The team members gathered for the introduction session headed by Dr. K.V.Anusuya, where we were introduced to FTC and got the briefing about this year's task of Rover Ruckus. Several points regarding the performance of PSG teams in past FTC competitions were provided and guidelines were shared in this session.

This team is of 6 members headed 4 departments of two members from Electronics and Communication Engineering and two are from Mechanical, one from Robotics and Automation Engineering and One from Computer Science Engineering. We made effort in each part of our BOT preparation. Each member has the contribution of Development and Implementation and Analysis. This bot is named PHOENIX and has unique design and performance.

1.2. HARDWARE SCOPE

Our preparation of BOT started with Mechanical Design with the small description of motors that are to be used for the BOT. We did the Mechanical part of the Phoenix with testing each part many times with modification. Using various motors and Servos, adhering to Game rules, we made Mechanical design and Implementation of the BOT.

The main Scope of our Design of the BOT is, the bot can be operated with more speed and also when compared to previous designs, our bot can be lifted up to half of the plane locking with handle through L-clamp. Then, lifting capability of nearly 2 to 3 materials at a time and placing in the Cargo Hold. We optimised the design of the bot with many modifications in the mechanical design.

1.3. SOFTWARE SCOPE

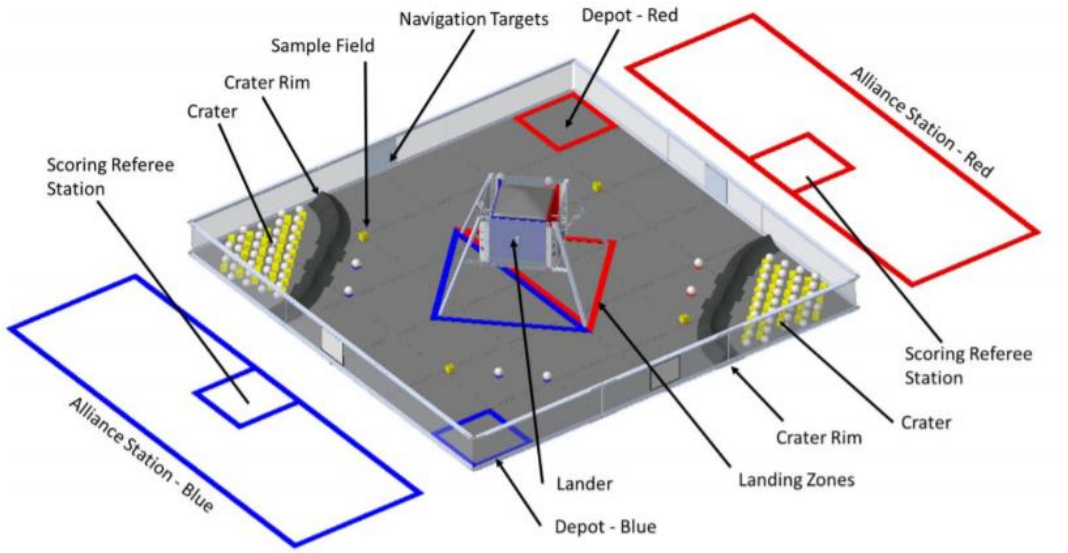
In the software part of the BOT, we started with the study of Beagle Bone that to be used for the project. Then the implementations and the testing of DC motors and

Servo Control is implemented and tested. During the manual control period of the game, the bot is controlled using a wireless game joystick. We used EVDEV library and its functions for interfacing the beaglebone black with joystick. For autonomous period, our main aim in the Software part is to finish the Cube Detection with the Time slice. The cube detection is made with the combination of the Colour detection and shape detection (using contours – edge approximation). The software scope of our Phoenix is, the bot can detect multiple materials and track them in any noisy environment. We used time span and optimised the implementation from 30 seconds to 4 seconds. Using a wireless joystick, we can control the Phoenix.

1.4. GAME DESCRIPTION

(From the official FTC website)

Matches are played on a Playing Field initially set up as illustrated in the figure below.



Two Alliances – one “Red” and one “Blue,” composed of two Teams each – compete in each Match. The objective of the game is to attain a higher Score than the opposing Alliance by Scoring materials into the lander, lifting the bot up to half of the plane, Cube tracking, end game task, performing Autonomous tasks, and navigating to specific parts of the Playing Field. The Scoring Elements for the game are 86 gold scoring elements, 52 silver scoring elements and also two silver scoring minerals and one gold mineral placed separately together for autonomous task (Tracking the gold mineral out of the three minerals).

The game is played in two distinct periods: Autonomous and Driver-Controlled. The Match starts with a 30-second Autonomous Period in which Robots operate using pre-programmed instructions only. Alliances earn points by

1. Landing of bot from lander
2. Sampling Gold Mineral
3. Placing the Team marker in Depot
4. Parking on Crater rim

The two-minute Driver-Controlled Period follows the Autonomous Period. During this period, Teams earn points for their Alliance by:

1. Placing gold and silver minerals into lander.
2. Minerals Scored into the Depot earn two (2) points each. Minerals removed from the Depot deduct two (2) points each.
3. Gold Scored into the Gold Cargo Hold on the Lander
4. Silver Scored into the Silver Cargo Hold on the Lander

The final 30 seconds of the Driver-Controlled Period is called the End Game. In addition to the previously listed Driver-Controlled Period Scoring activities, Alliances earn points by:

1. Robots Latched
2. Robots Parked In any Crater
3. Robots Parked Completely In any Crater.

ACTIVITY	POINTS
AUTONOMOUS PERIOD	
1. Landing of bot from lander	30
2. Sampling Gold Mineral	25
3. Placing the Team marker in Depot	15
4. Parking on Crater rim	10

DRIVER CONTROL PERIOD	
1. Gold Mineral in Gold Cargo Hold	5
2. Silver Mineral in Silver Cargo Hold	5
3. Any Mineral in Depot	2
END GAME	
1. Robots Latched.	50
2. Robots Parked Partially Crater	15
3. Robots Parked Completely In Crater.	25

1.5. PLAYING FIELD DEVELOPMENT

Discussions on the development of common playing field were conducted jointly by both the PSG teams. The subtasks for the development of the field were allotted among the members of both the teams. The engineering practices laboratory was used as a venue for the field. Purchases of various materials for the construction of the Lander, silver scoring materials and gold scoring materials and crater rim were made. The members of both teams used these materials to construct aforementioned field components.

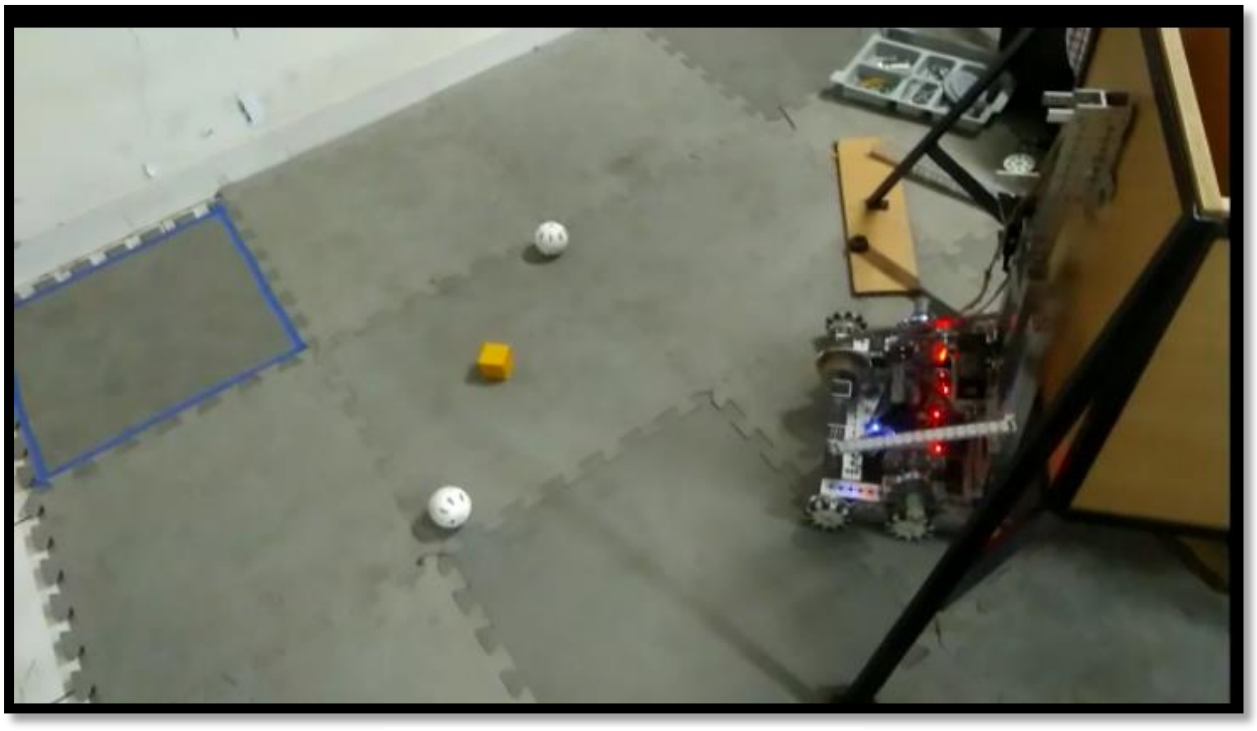


Fig 1. Bot handled to the Crater

Fig 2. Layout of scoring minerals with lander

Fig 3. Lander Setup

2. SOFTWARE

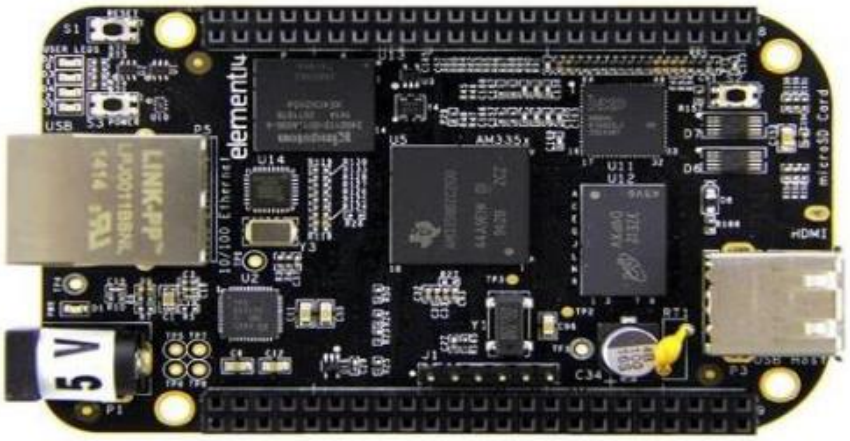
2.1.SOFTWARE SCOPE

The scope of the software component of the rover rockus bot is:

1. A real time operating system running on a micro-controller
2. Code to manage the various mechanical activities of the bot
3. Image processing to detect the minerals even in noisy environment.
4. Code for autonomous operation
5. Manual control interfacing with bot.

2.2. TOOLS AND LIBRARIES

- **Microcontroller: Beagle Bone Black**



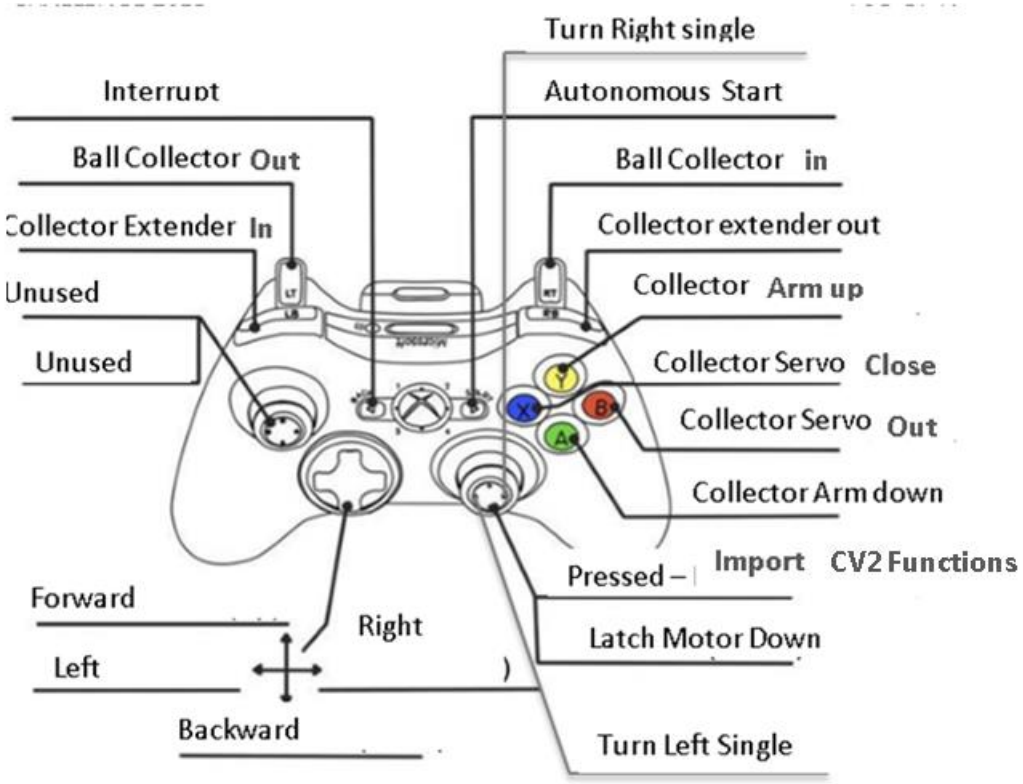
1. The Beagle Bone Black is an affordable single-board credit card sized computer. It has a powerful ARM Cortex A8 CPU, a full Linux OS, and allows for easy access to external sensors.
2. It is an open platform targeting students and hobbyists who want hardware focussed alternative to the Raspberry PI.
3. Default IP address for Beagle Bone Black: 192.168.7.2

- Operating System : Debian 9.5 IoT
- IDE : Cloud9
- Scripting Language : Python 2.7
- Image processing libraries: OpenCV, Python Imaging Library (PIL), Numpy.
- Joystick input handling library : evdev

2.3. MANUAL CONTROL DESIGN

During the manual control period of the game, the bot is controlled using a wireless game joystick. The various functions for movement and activation of mechanisms (Refer chapter 3 for more information) are mapped to buttons on the joystick as illustrated below:

(Note: Microsoft Xbox controller used for reference only)



2.4. AUTONOMOUS PERIOD – TACTICAL OVERVIEW

Based on the game description, the Autonomous period can be broken down into 4 main functions:

1. Landing of bot from lander
2. Sampling Gold Mineral
3. Placing the Team marker in Depot
4. Parking on Crater rim

Keeping these functions in mind, we planned several strategies of play, each of which having its own set of pros and cons which we considered before finalizing on one.

In order to landing the bot from the lander, we used the functions to perform landing the base of the bot down, then to unlock the connection of bot from the lander. After landing, the bot use the function forward, turning left and right for necessary autonomous operations.

For, sampling gold mineral, our bot runs a code snippet that runs the OpenCV library. This code is made to identify the gold mineral out of three minerals using the colour and shape detection of cube.

The autonomous code contains several functions to perform various mechanical actions such as moving forward, turning left or right, reversing and activating the collection mechanism, the lift mechanism and the jewel displacement mechanism (Discussed further in 3.3, 3.4 and 3.5 respectively).

Each autonomous strategy implementation had to be replicated 4 times – each modified for a different starting position. The 4 possible starting positions are:

1. Red Alliance position 1
2. Red Alliance position 2
3. Blue Alliance position 1
4. Blue Alliance position 2

Where “position 1” refers to the bot from Lander closer to the crater rim.

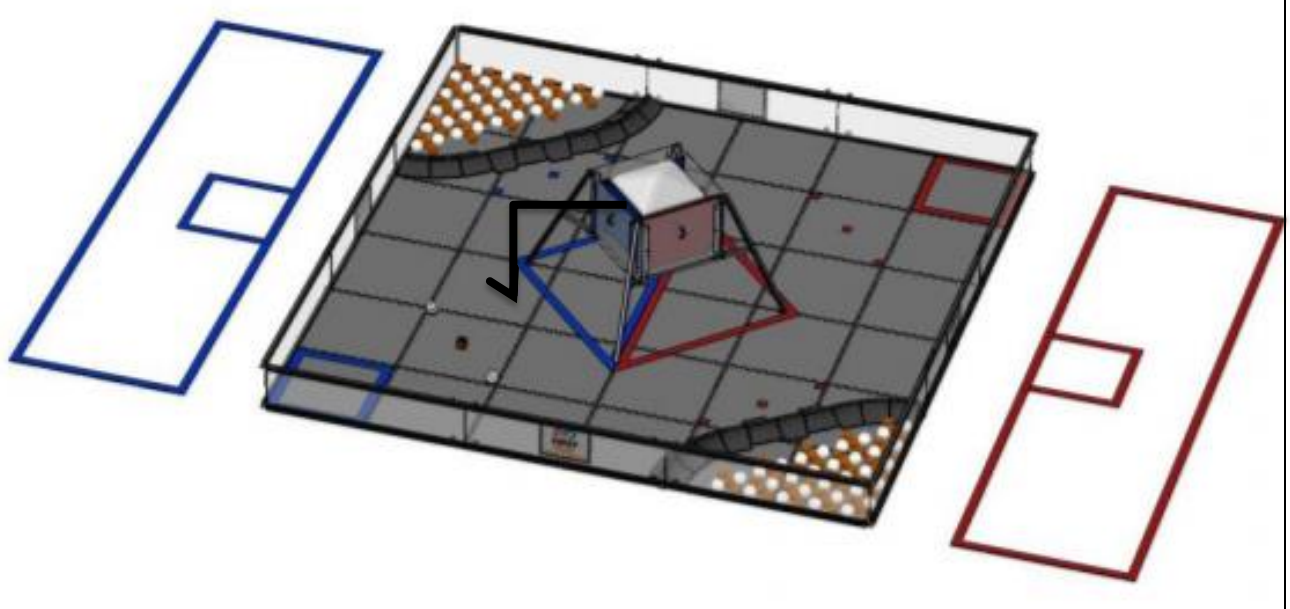
Where “position 2” refers to the bot from lander closer to the depot.

2.5. AUTONOMOUS PERIOD – STRATEGIES

As mentioned in section 2.4, there are 4 main functions in the autonomous section of the game. We formed 6 strategies that utilize different combinations of the above functions.

There are different strategies for the autonomous period for scoring points. The strategies are made to score points in planned way and also for safe scoring.

1. STRATEGY 1: THE “SAFE” STRATEGY



In this strategy the bot lands down from the lander.

PROS:

- ⊕ Guarantees 30 points
- ⊕ Maximum scoring of points
- ⊕ Almost no room for error

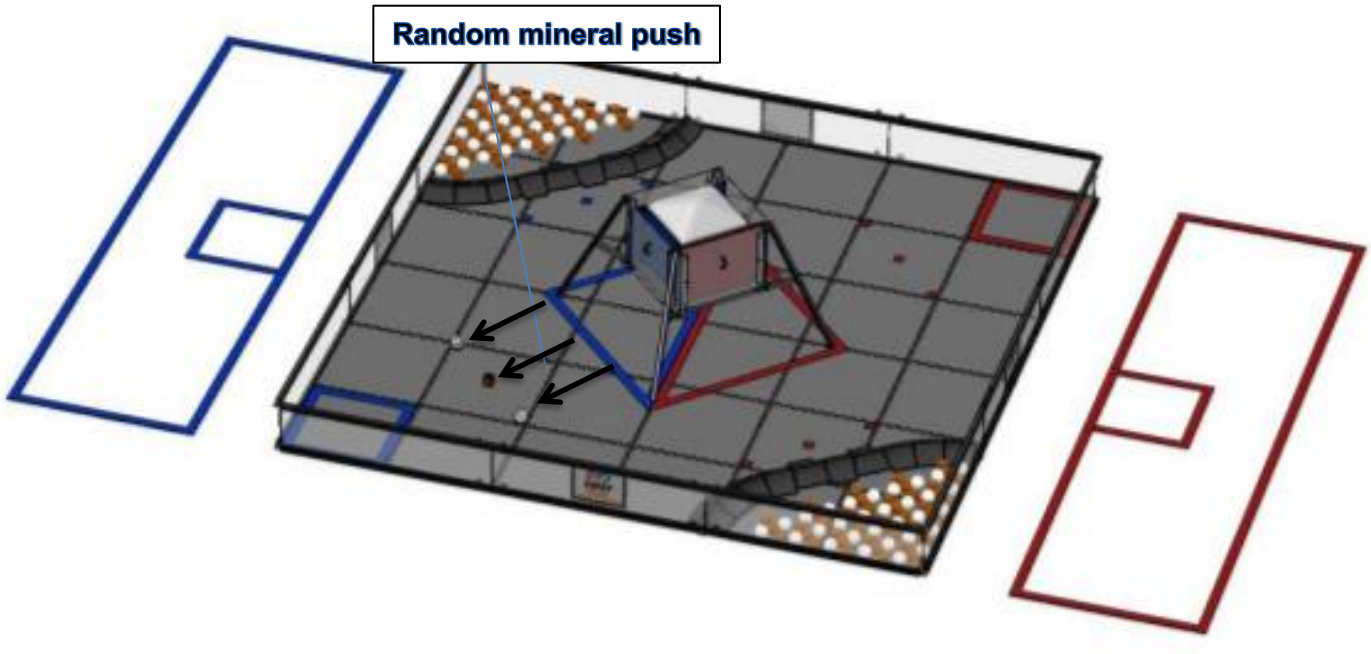
CONS:

- Only 30 points
- Chances of connections miss lead.
- Not very creative
- Time gets wasted

Task	Points Scored
Landing of Bot from Lander	**30**

2. STRATEGY 2: THE "GOLD LOTTERY" STRATEGY

In this strategy the bot activates the Cube mechanism (Refer 3.5) at the



starting position. It then moves forward and push the mineral in the middle as per implementation. Then it moves right towards the crater rim and simply parks there.

PROS:

- ⊕ Guarantees 25 points.
- ⊕ Easy to implement

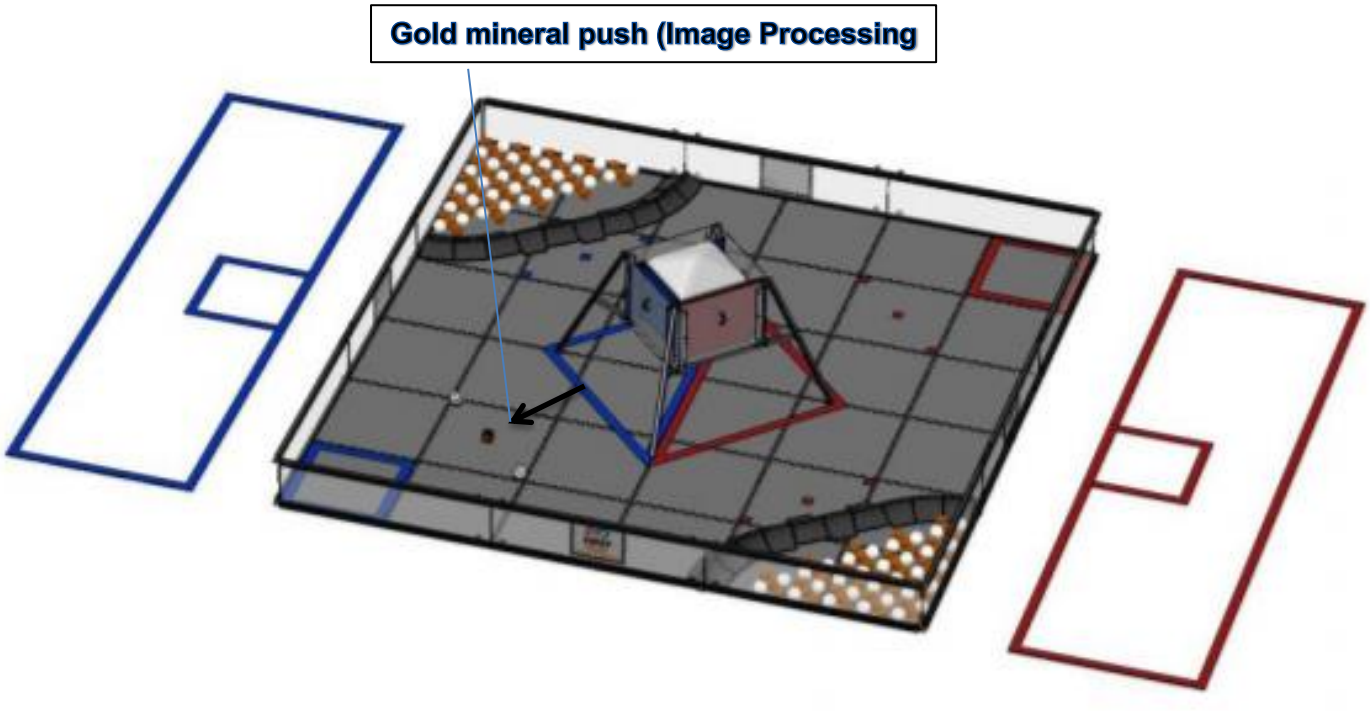
CONS:

- - No points scored pushing of wrong mineral.
- - Time gets wasted

Task	Points Scored
Landing of Bot from Lander	30
Sampling Gold Mineral	25
Total	**45**

3. STRATEGY 3: THE "GOLD EXPERT" STRATEGY

In this strategy the bot activates the cube detection mechanism at the starting position. It then uses image processing to determine which side the Gold mineral is



using object detection with contours. It then moves forward/backward, knocking over the Gold mineral. It pushes the mineral towards the depot and then simply moves forward towards the crater rim and parks on it.

PROS:

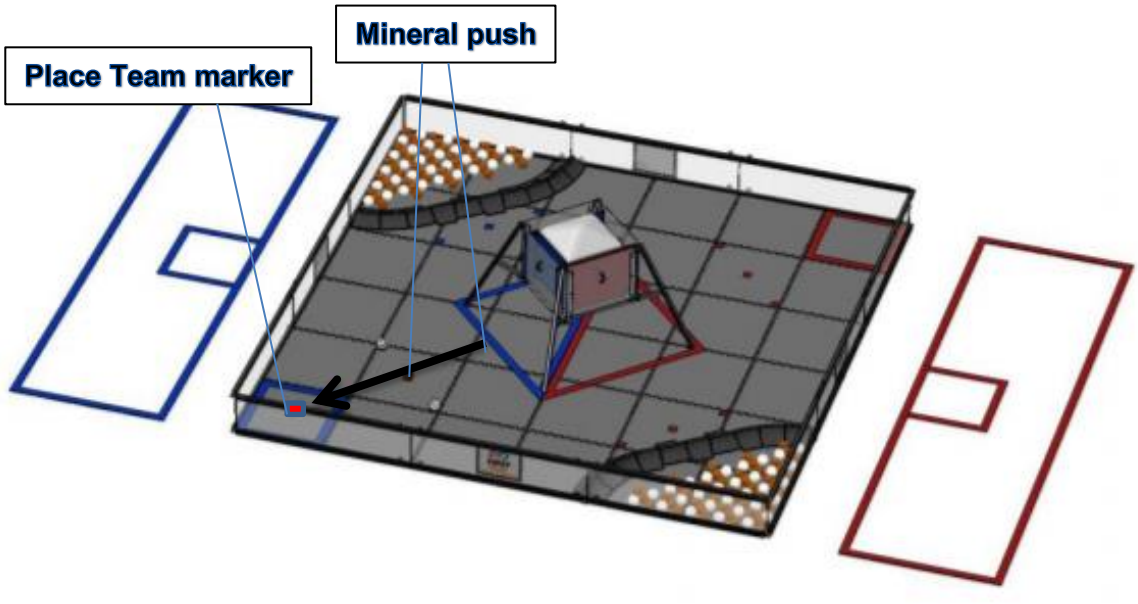
- ⊕ Guarantees 25 points.
- ⊕ Perfectly tracks the gold mineral

CONS:

- Moderate risk of error
- Risk of wrong mineral detection due to background.
- Time gets wasted

Task	Points Scored
Sampling Gold Mineral	25
Landing Bot from Lander	30
Total	**45**

4. STRATEGY 4: TEAM MARKER STRATEGY



After pushing the gold mineral, the bot move forward towards depot and places the team marker in depot. Then, it move right towards crater rim and parks on it.

PROS:

- ⊕ Guarantees 15 points.
- ⊕ Chance of more score.

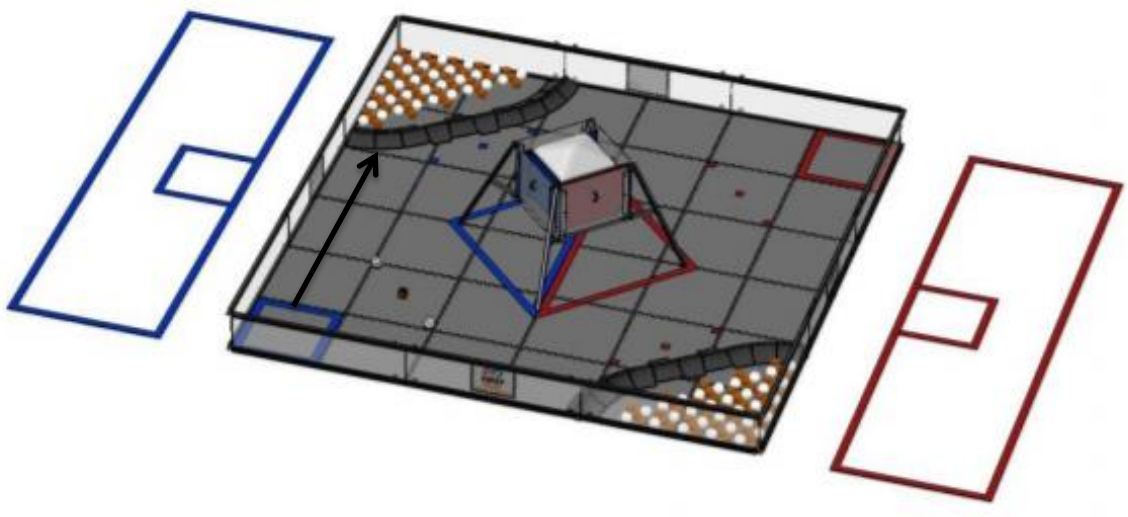
CONS:

- Moderate risk of error
- Time gets wasted

Task	Points Scored
Placing Team Marker	15
Sampling Gold Mineral	25
Landing Bot from Lander	30
Total	**70**

5. STRATEGY 5: THE "I'M FEELING LUCKY" STRATEGY

The bot from depot moves forward towards the crater rim and parks on it. This ends the tasks of autonomous period.



PROS:

- ⊕ Guarantees 10 points.
- ⊕ Chance of more score.
- ⊕ Easy to implement.

CONS:

- ⊕ Chances of Hardware connection failure.
- ⊕ Risk of error.

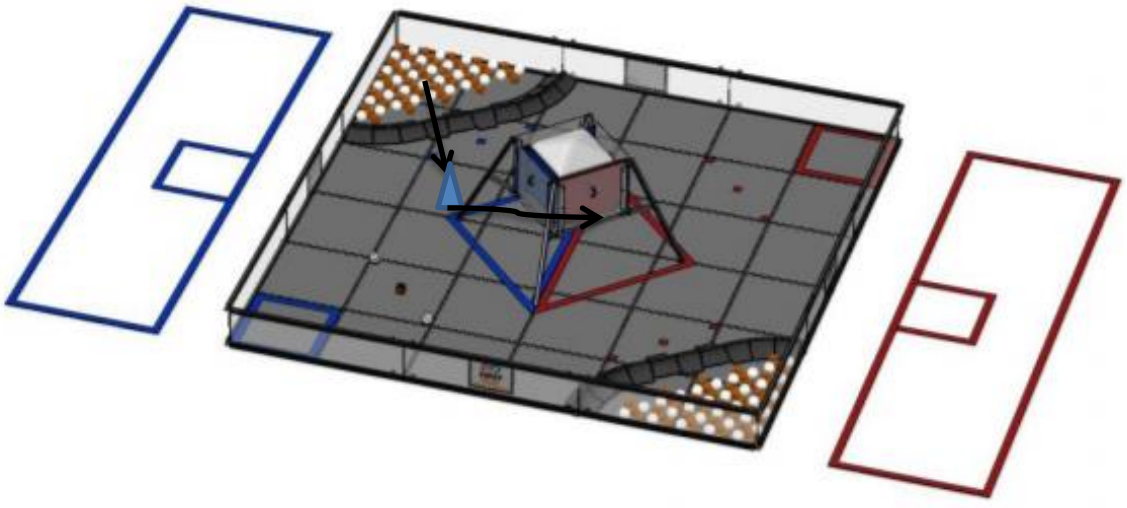
Task	Points Scored
Parking on Crater Rim	10
Placing Team Marker	15
Sampling Gold Mineral	25
Landing Bot from Lander	30
Total	**80**

2.6. MANUAL PERIOD – TACTICAL OVERVIEW

Based on the game description, the Autonomous period can be broken down into 4 main functions:

1. Placing Gold Mineral in Cargo Hold
2. Placing Silver Mineral in Silver Cargo Hold
3. Placing Any Mineral in Depot

Keeping these functions in mind, we planned several strategies of play, each of which having its own set of pros and cons which we considered before finalizing on one.



For placing minerals from crater to the gold cargo hold, the bot moves towards the crater and prolong it's handle into the crater and pulls the minerals into it.

For placing gold minerals into the gold cargo hold, we designed the bot such that it have two ways connecting to other side to which we can put the mineral in it. The two ways are designed with the measure that one way can allow only the gold minerals. After pulling the gold minerals into bot, the bot moves towards the cargo hold and pushes the gold mineral into it.

For placing Silver mineral into the silver cargo hold, the silver mineral will be pulled from the crater to the bot through one of the two ways that allows the silver mineral comparing size.

For placing the mineral in the depot, the robot controlled manually will move forward and pushes the mineral into the depot.

2.7. END GAME PERIOD – TACTICAL OVERVIEW

The end game period is the last 30 second period of the game in which the bot will park in the zone as mentioned in the task. The end game period consists of three tasks that the bot uses one of the two tasks to park during the last 30 second period.

END GAME	
1. Robots Latched Back onto Lander	50
2. Robots Parked In Crater	15
3. Robots Parked Completely In Crater.	25

During the end game period, the bot can continue with placing minerals into the cargo hold else the bot will park using either of the three tasks.

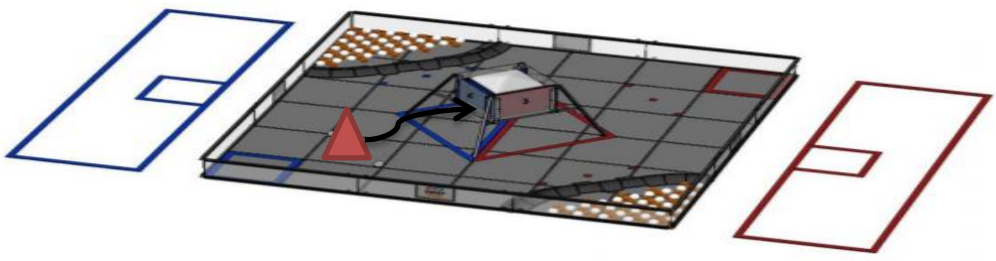
For the robot to be latched, we designed the robot and made implementations so that during the end game period the bot will come near the lander lock itself to the handle of the lander using a L clamp and then it can be latched back.

The other way for scoring points in end game is, “parking in crater”. For this task, the bot moves towards the crater and parks on the crater rim (Edge of the crater). This is the easiest task.

The bot can move inside the crater contains the minerals and parks there. This is the second easiest task for the bots which is in perfect condition to score good points.

2.8. END GAME PERIOD – STRATEGIES

1. **STRATEGY 1: HAPPY END STRATEGY**



PROS:

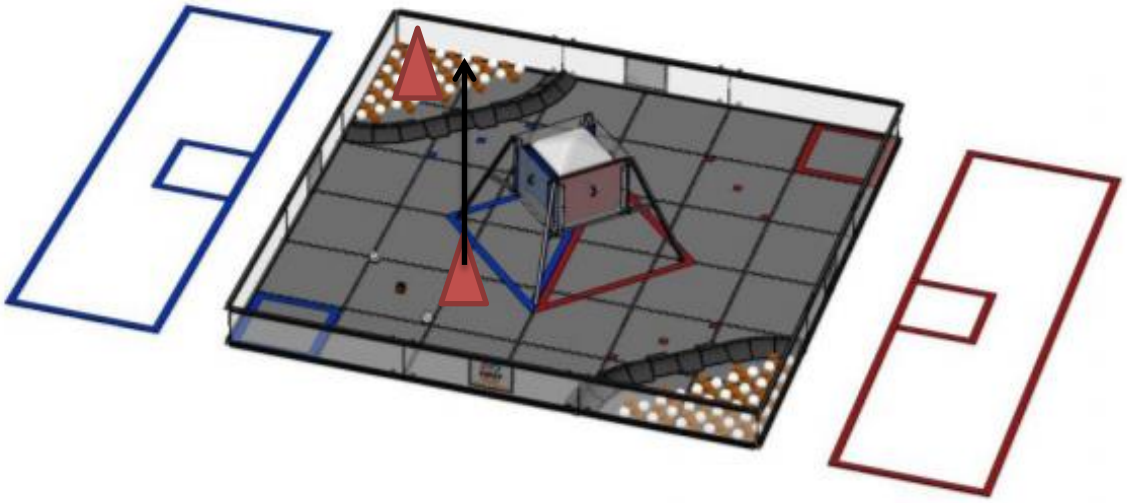
⊕ Guarantees 50 points.
⊕ Chance of more score.
⊕ Easy to implement.

CONS:

- Chances of Hardware connection failure.
- Risk of error.

Task	Points Scored
Robot Latched	50

2. STRATEGY 2: SAFE END STRATEGY



PROS:

⊕ Guarantees 25 points.
⊕ Chance of more score.
⊕ Easy to implement.

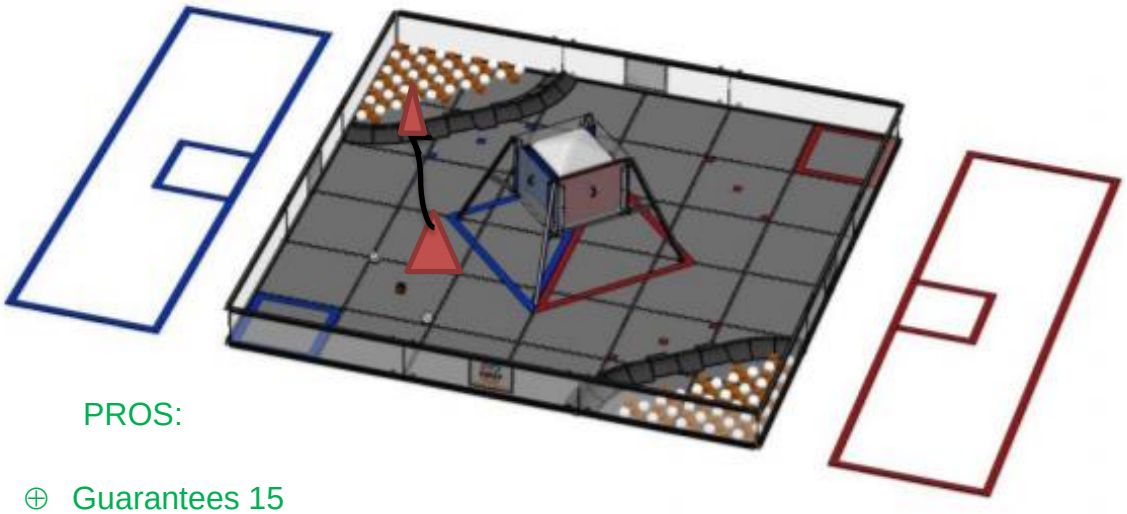
CONS:

- Only 25 points
- Chances of connections miss lead.

Task	Points Scored
Parking on Crater Rim	15

3. STRATEGY 3: SIMPLE END STRATEGY

The bot parks on the edge of the crater rim during this end game period. This task also a safe task and simple task for bot to move forward from the current position and park on crater rim.



PROS:

- ⊕ Guarantees 15 points.
- ⊕ Simple and Safe
- ⊕ Easy to implement.

CONS:

- Only 15 points
- Time gets wasted

Task	Points Scored
Parking into the Crater	25

2.9. AUTONOMOUS PERIOD – PSEUDOCODE

SAFE STRATEGY

```
def autonomous_One():
        print('Autonomous Strategy 1 Running\n\n')
        latch_Down
        print('Bot Lowered')
        motor_Stop()
        backward_Run
         print('Bot Grounded')
        motor_Stop()
        time.sleep
         latch_Servo("P9_14", 90)
        print('Servo Unlocked')
         time.sleep
         #marker servo test
        marker_Servo("P9_16",90)
        print('Marker_Released')
        motor_Stop()
        time.sleep
         marker_Servo("P9_16",0)
        print('Servo_closed')
         motor_Stop()
```

THE GOLD LOTERY STRATEGY:

```
Autonomous_Lotery(Position,  Alliance)
{
        Time = set_Timer
        //Set movement direction to reverse
        while(Time > 0) do
        forward_Run
        motor_Stop()
        time.sleep
        run_Motors(Dir);
        delay(time_to_safezone);
        //time_to_safezone is variable
        stop_Motors();
        END
 }
```

In this Strategy, the bot will be implemented so that after getting down from the lander, the bot moves forward and takes the middle mineral towards the Depot. If the mineral is gold, then our team score points. Else, the points will not be given.

THE "GOLD EXPERT" STRATEGY – GOLD ON CENTER

```
Autonomous_Gold(Position,  Alliance)
        direction = image_Process(0)
        if (direction == centre):
                print('Gold is on the Centre')
                ##Timing for the battery voltage
                forward_Run
                motor_Stop()
                time.sleep
                turn_Right_All_Wheels
                motor_Stop()
                time.sleep
                marker_Servo("P9_16",90)
                print('Marker_Released')
                motor_Stop()
                time.sleep
                marker_Servo("P9_16",0)
                print('Servo_closed')
                motor_Stop()
                backward_Run
                motor_Stop()
                turn_Left_All_Wheels(1)
                motor_Stop()
                time.sleep
```

THE “GOLD EXPERT” STRATEGY – GOLD ON CENTER

```
Autonomous_Gold(Position,  Alliance)
        direction = image_Process(0)
        if (Gold on right):
                print(‘Direction == right')
                forward_Run
                motor_Stop()
                time.sleep
                turn_Left_All_Wheels
                motor_Stop()
                time.sleep
                forward_Run
                motor_Stop()
                time.sleep
                turn_Right_All_Wheels(0.6)
                motor_Stop()
                time.sleep
                forward_Run(1)
                motor  Stop()
```

THE "GOLD EXPERT" STRATEGY – GOLD ON LEFT

```
Autonomous_Gold(Position,  Alliance)
        direction = image_Process(0)
        if (direction == left):
                print('Gold is on the Left')
                forward_Run
                motor_Stop()
                time.sleep(0.5)
                turn_Right_All_Wheels
                motor_Stop()
                time.sleep
                forward_Run
                motor_Stop()
                time.sleep
                turn_Left_All_Wheels
                motor_Stop()
                time.sleep
                forward_Run
                motor_Stop()
                time.sleep
                turn_Right_All_Wheels
                motor_Stop()
```

THE "I'M FEELING LUCKY" STRATEGY

```
Autonomous_End(Position,  Alliance) :
        turn_Left_All_Wheels
        motor_Stop()
        time.sleep
        forward_Run
        motor_Stop()
        time.sleep
        turn_Right_All_Wheels
        motor_Stop()
        time.sleep
        marker_Servo("P9_16",90)
        print('Marker_Released')
         time.sleep
         marker_Servo("P9_16",0)
        print('Servo_closed')
        time.sleep
        #backward_Run
        #motor_Stop()
        #turn_Left_All_Wheels(1)
        #motor_Stop()
        #time.sleep
 #forward_Run
```

3. IMAGE PROCESSING – FOCUS

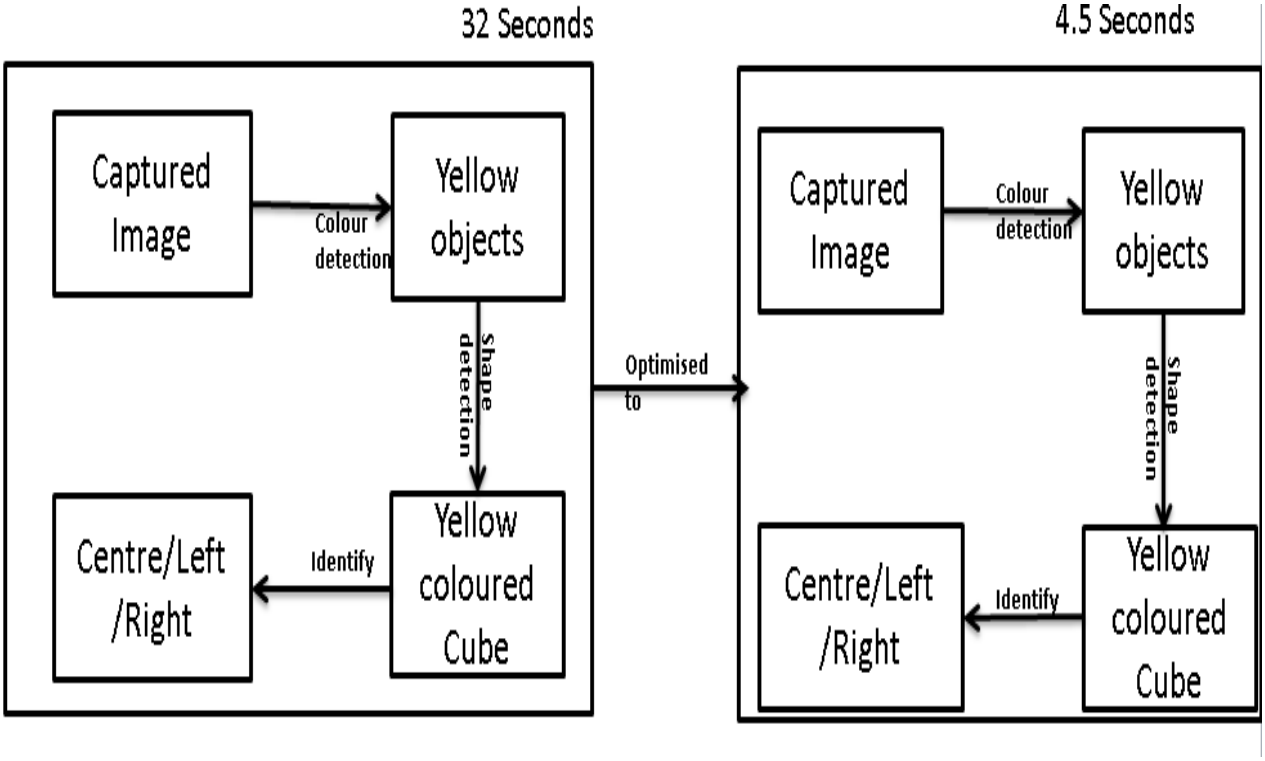
3.1. IDEA OF THE TASK:

- Capture the Front ground.
- Use Colour and Shape detection mechanism.
- Detect the cube.
- Tracking the cube in the mentioned directions.
- Optimise the Time Taken for the task.

3.2. OVERALL DESIGN:

Processing steps we made:

- Colour Detection mechanism for Yellow colour Detection.
- Shape detection mechanism to detect cube using Contours.
- Optimised the implementation to run the Autonomous period fast.



3.3. COLOUR DETECTION:

We used colour Detection mechanism to detect the yellow coloured objects for the captured image when our bot get down from the lander.

Software Libraries:

1. cv2
2. numpy

We used Gaussian blur() function in colour detection to detect the perfect yellow colour using the Binary to HSV conversions of applied frequency for yellow colour.

Processing Steps:

- Bot captures the front image.
- The captured image made blurred using Gaussian Blur().
- The blurred image then converted to HSV.
- Using Numpy array, the frequencies of the yellow colour is entered.
- Using the above conditions, the output image will be displayed.

Implementation:

```
// Importing libraries
import cv2
import numpy as np
blurValue = 41 // Assign blur Value for Gaussian Blur()
camera = cv2.VideoCapture(0)
while camera.isOpened():
        (grabbed, frame) = camera.read()
        blur = cv2. GaussianBlur(frame, ((blurValue, blurValue)), 0)
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV) // Conversion of BGR to HSV
        // Providing Yellow coloured Frequencies
        lower = [18,50,50]
         upper = [35,255,255]
         lower = np.array(lower, dtype = "float32")
         upper = np.array(upper, dtype = "float32")
        mask = cv2.inRange(hsv, lower, upper)
          output = cv2.bitwise_and(frame, hsv, mask = mask)
        cv2.imshow("output", output) // Final Output Image
```

Execution of Colour Detection:

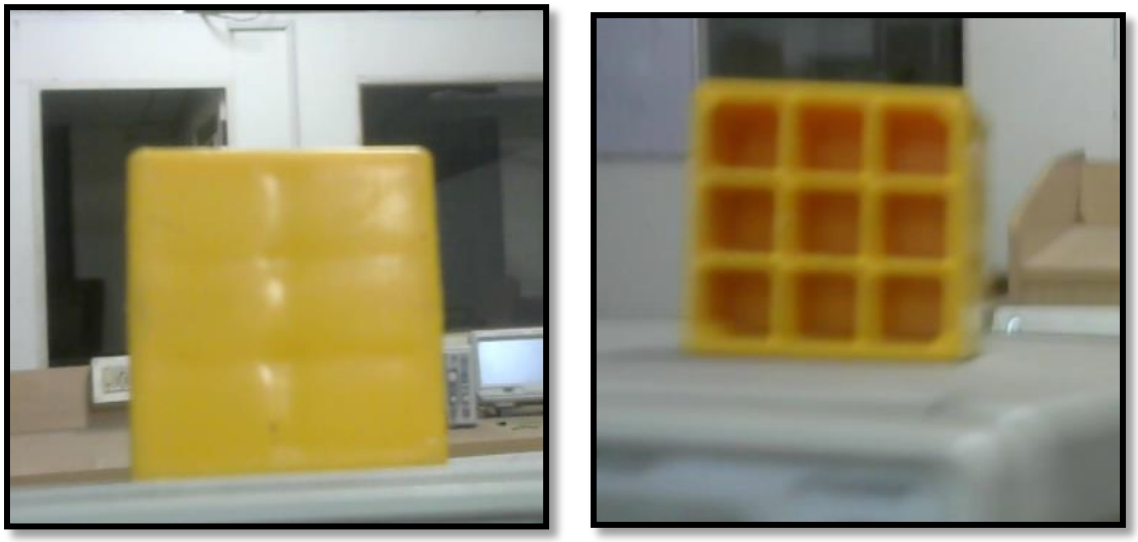


Fig 1: Captured Image

Fig 2: Conversion to Binary

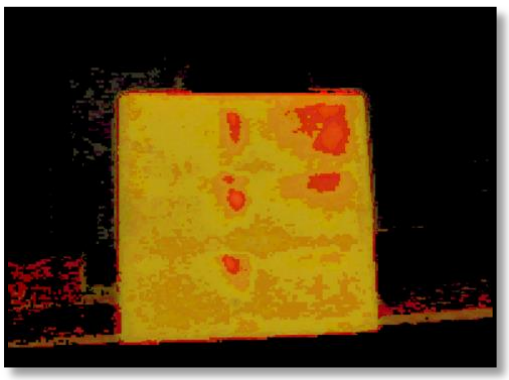


Fig 3: Final Output Image

We could’nt use only colour detection mechanism for the task because it will not give the perfect output for cube detection instead it will detect other objects near which are in yellow coloured frequencies.

3.4. SHAPE DETECTION:

We added shape detection to the colour detection to provide perfect output of detecting the Gold coloured mineral with out any background effect.

Software Libraries:

1. Opencv
2. Numpy

We used Contours to detect the shape using the end points formation on yellow coloured objects. We provided the condition of less than 16 end points for the gold mineral detection.

Processing Steps:

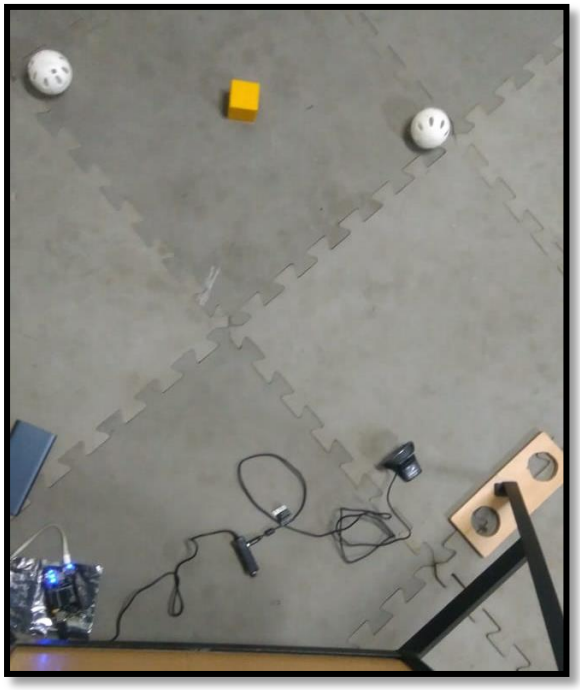
1. The contours are formed on the region where yellow color present.
2. The condition is given as if the approximation of contours in region is less than 12, then it will be declared as Cube. Else, the implementation of this task will be stopped.
3. If the cube is present in the image, then based on the pixel range in image, the place of the cube is decided (Right or Left or Centre)
4. If the cube is present in left side of the minerals (-1), then the bot moves towards left from the lander, push the cube towards the depot.
5. If the cube is present in right side of the other minerals(+1), the bot move towards right side, pushes the cube forward left and moves towards the depot.

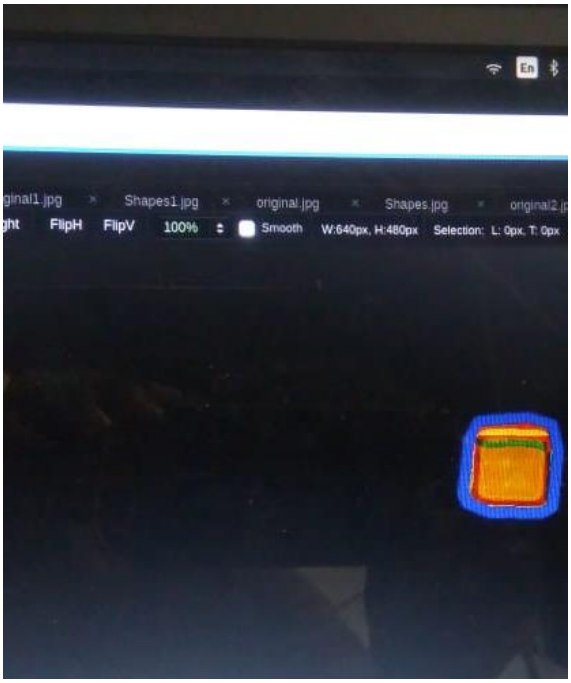
6. If the cube is in centre (0), the bot moves forward, pushes the cube towards the depot.
7. The whole task will be taken in 5 seconds during the Autonomous period.

Implementation:

```
import cv2
import numpy as np
camera = cv2.VideoCapture(1)
camera.set(100,200)
while camera.isOpened():
   ret, frame = camera.read()
   gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
   _, threshold = cv2.threshold(gray, 160, 255, cv2.THRESH_BINARY)
   contours, h = cv2.findContours(threshold, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
  font = cv2.FONT_HERSHEY_COMPLEX
 for cnt in contours:
      approx = cv2.approxPolyDP(cnt, 0.01*cv2.arcLength(cnt, True), True)
      cv2.drawContours(frame, [approx], 0, (0), 5)
      x = approx.ravel()[0]
      y = approx.ravel()[1]
      if len(approx)==4 :
         cv2.putText(frame, "Rectangle", (x, y), font, 1, (0))
      elif len(approx)>4:
         cv2.putText(frame, "Cube", (x, y), font, 1, (0))
   cv2.imshow("shapes", frame)
   cv2.imshow("Threshold", threshold)
   cv2.waitKey(0)
   cv2.destroyAllWindows()
```

Output:





3.5. TIME STAMP:

Initial Time Stamp:

The initial time stamp for execution of this task taken 32 seconds out of other autonomous tasks.

Problem with Initial Time stamp:

The total Autonomous period will be 30 second period. But, detecting the gold mineral take 32 seconds which exceed autonomous period time and the bot can't perform other autonomous tasks.

```
// Color Detection
import cv2
import numpy as np
def image_Process(ret):
    try:
        camera = VideoCapture(0)
        ret, frame = camera.read()
        blur = GaussianBlur(frame, ((blurValue, blurValue)), 0)
        hsv = cvtColor(blur, COLOR_BGR2HSV)
        lower = [18,50,50]
        upper = [35,255,255]
        mask = inRange(hsv, lower, upper) //Find yellow colour
                        return 1
```

```
// Shape detection
_,contours, hierarchy = findContours(mask, RETR_TREE, CHAIN_APPROX_SIMPLE)
        for cnt in contours:
                        approx = approxPolyDP(cnt, 0.01*arcLength(cnt, True), True)
            drawContours(mask, [approx], 0, (0), 5)
                        x = approx.ravel()[0]
                        y = approx.ravel()[1]
                        if len(approx)<10 :
                                    approx = approxPolyDP(cnt, 0.01*arcLength(cnt, True), True)
            drawContours(output, [approx], 0, (100), 10)
                                    print(x)
            else:
            drawContours(output, [approx], 0, (100), 10)
                        break
                        if x>=0 and x<=213 :
                        return -1
                        elif x>213 and x<=427:
                        return 0
                        elif x>427 and x<=640:
                        return 1
            except Exception as e:
            print(e)
            print('\n Process abrupted!!')
```

The above execution of the code takes 32 second time period due to the importing functions.

It took 25 seconds for importing libraries and 7 seconds for the execution.

To optimise the time of the task, we imported necessary functions from opencv and numpy libraries. After changing the implementation, only 7 seconds took for importing libraries and 5 seconds took to finish task.

To make the execution more fast, we made the importing functions to run in starting of autonomous period and execute the task after that. So, finally the task completed within 4 second period.

```
From cv2 import VideoCapture, GaussianBlur, COLOR_BGR2HSV, cvtColor, bitwise, inRange, imwrite,
findContours,  approxPolyDP, drawContours, arcLength, CHAIN_APPROX_SIMPLE, RETR_TREE,
waitkey(),destroyAllWindows

From numpy import array

// Color Detection

import cv2

import numpy as np

def image_Process(ret):

    try:

        camera = VideoCapture(0)

        ret, frame = camera.read()

        blur = GaussianBlur(frame, ((blurValue, blurValue)), 0)

        hsv = cvtColor(blur, COLOR_BGR2HSV)

        lower = [18,50,50]

        upper = [35,255,255]

        mask = inRange(hsv, lower, upper) //Find yellow colour

                        return 1
```

```
_,contours, hierarchy = findContours(mask, RETR_TREE, CHAIN_APPROX_SIMPLE)
        for cnt in contours:
                        approx = approxPolyDP(cnt, 0.01*arcLength(cnt, True), True)
            drawContours(mask, [approx], 0, (0), 5)
                        x = approx.ravel()[0]
                        y = approx.ravel()[1]
                        if len(approx)<10 :
                                    approx = approxPolyDP(cnt, 0.01*arcLength(cnt, True), True)
            drawContours(output, [approx], 0, (100), 10)
            else:
            drawContours(output, [approx], 0, (100), 10)
                        // cube location
            except Exception as e:
            print('\n Process abrupted!!')
```

HARDWARE

4.1 Introduction to scope of work

- Began building the robot based on hand drawn ideas. Developed the CAD model from the design of the bot.
- Started Design of the Bot with the base testing and continued with the modifications and testing in each stage of Development.
- We decided to share the Bot work among our team mates. The team partitioned with Three for Mechanical design and two for electrical and circuit connections and one for Image processing and Software Development.
- By starting with the chassis that gave us something to base the planned attachments off. We measured the space and each sub-team explained to the rest what their attachment would require. After all the information was pooled, we worked together to decide the best placement for each attachment.

4.2 Various mechanisms used

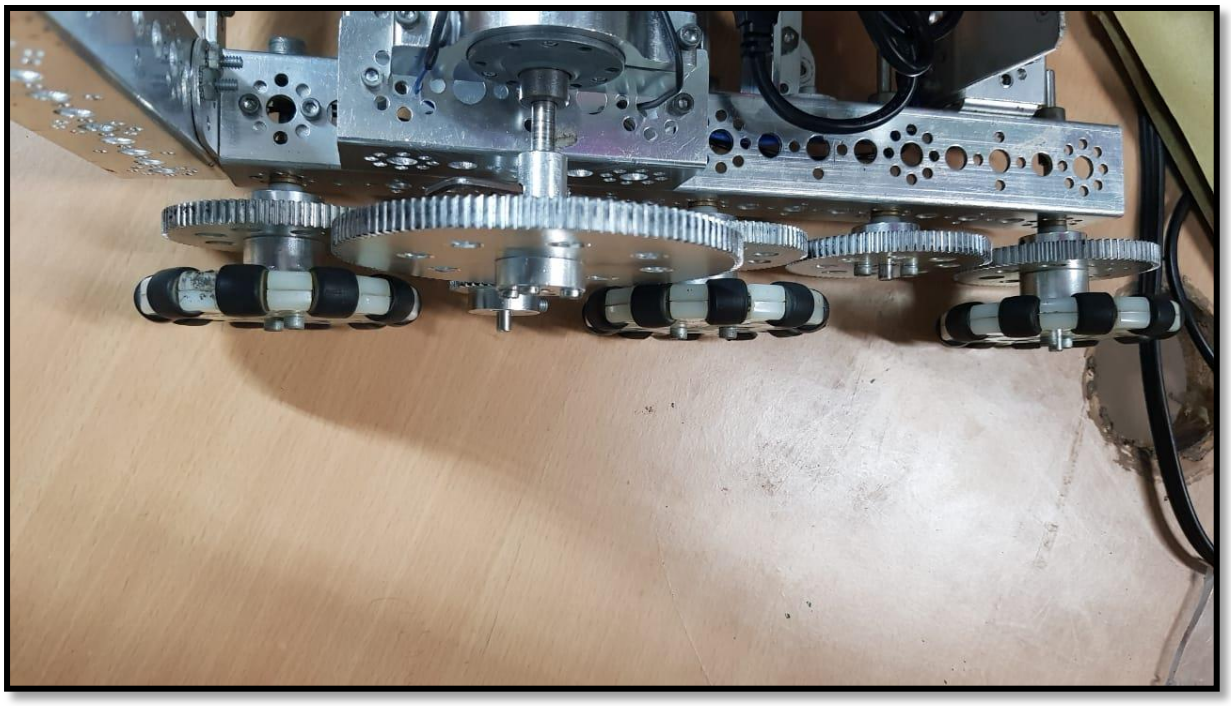
I. Drive train
II. Rope pulley mechanism
III. Extension arm
IV. Collection box

4.2.1 Drive train

A robot's drive train is the motors, wheels, axles, chains threads that are used to move the bot around.

- Two wheel drive
- Four wheel Drive
- West coast drive

All the above given types were applied and tested.

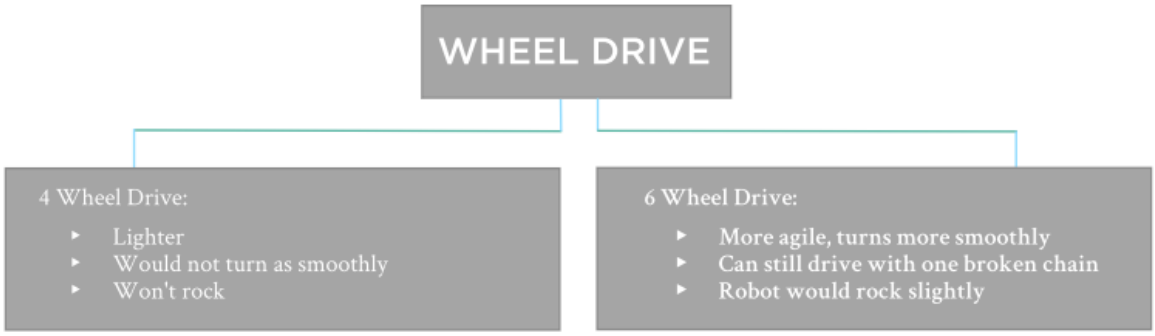


The 6 wheels had greater traction and rugged movement at all rough cases. As knowing in case of any movement required in the mineral filled corners, these would run through them.

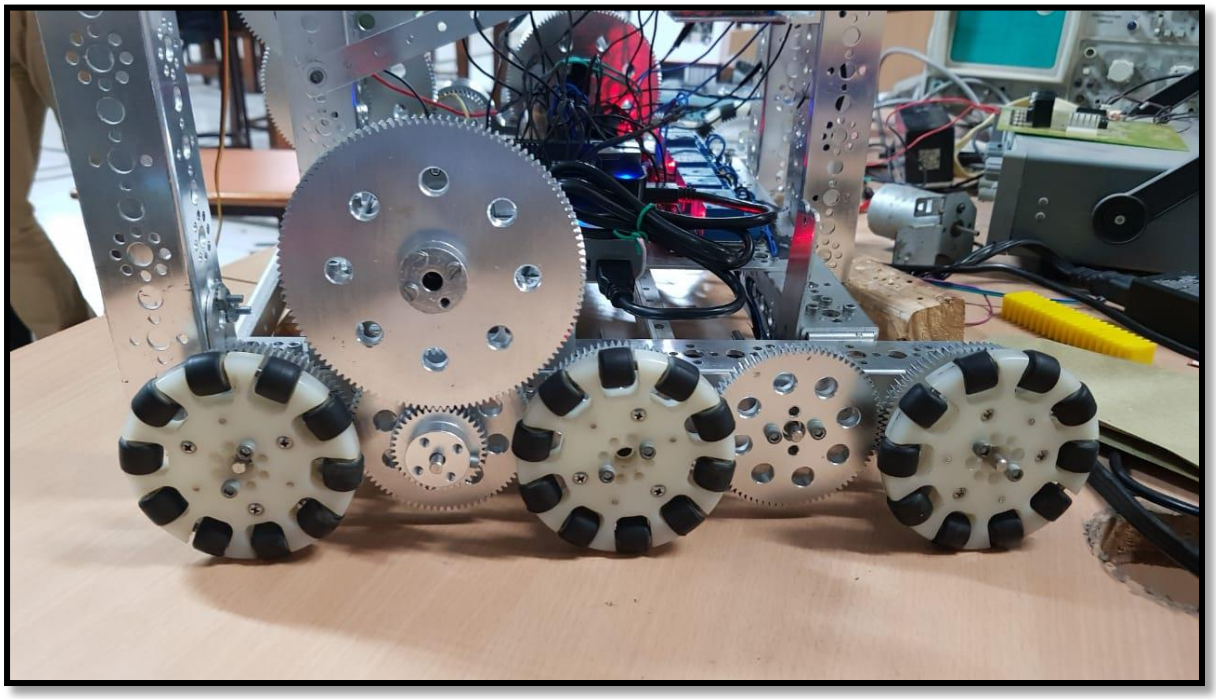
We found that 6 wheels were best suited and they were implemented.

Navigation:

- It uses a west-coast drive, driven by 2 motors placed by each side.
- The motor is connected to a 120 Teeth spur gear which is connected to another 30 Teeth spur gear which is again connected to a 60 Teeth spur gear in compound.

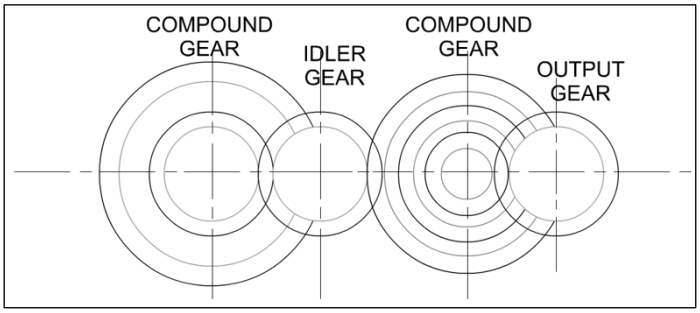


6 Wheel drive had a greater traction, so we went for it.



Omni wheels will make a easier drive in case of turns. Using the west coast drive, the bot has the highest turning radius.

Given that all wheels have a drive with motor; it makes bot to navigate easily through any terrain.

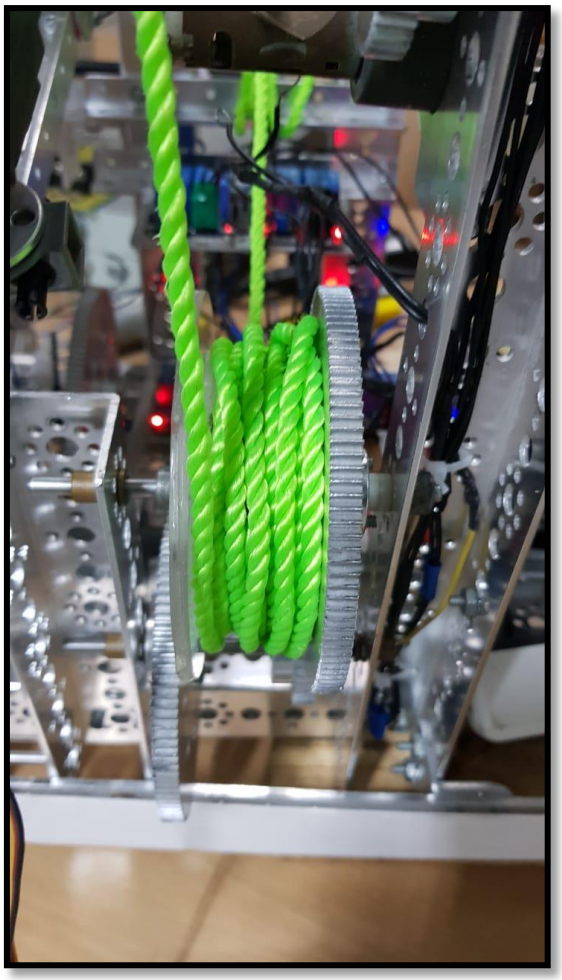


4.2.2 Rope pulley

Rope pulley	Gear system
• This can have more torque • Less space accommodated • Faster retraction	• Less torque to size ratio • Very large space even when compound gears are used • Slow retraction

So rope pulley was applied.

- A pulley type, double side rope attached beam will actuate to lower the bot from Lander to ground



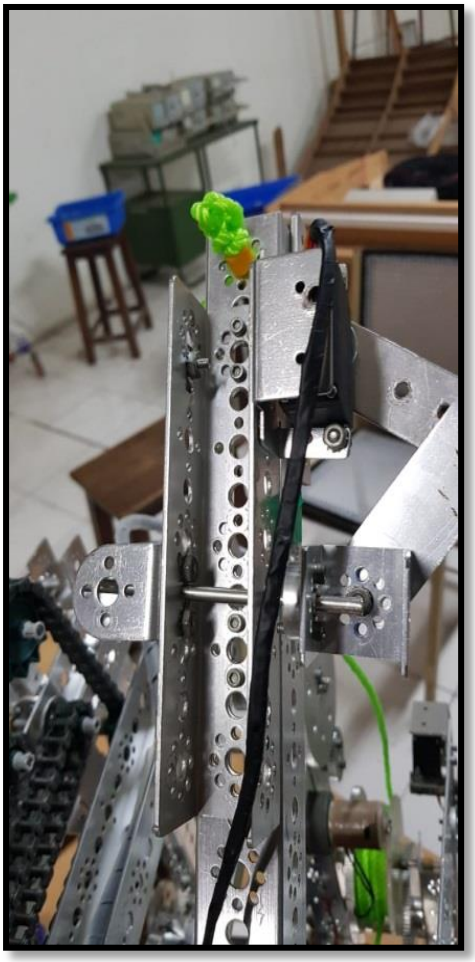
- At autonomous mode, the slider crank mechanism will remove from the lander clamp after landing on the ground.
- Then the image processing happens to detect the gold mineral and chose the strategy.
- This method can perform both landing down or hang up as the pulley rope is tied in both ends of the beam.

Slider crank mechanism

- A slider-crank linkage is a four-link mechanism with three revolute joints and one prismatic, or sliding joint.

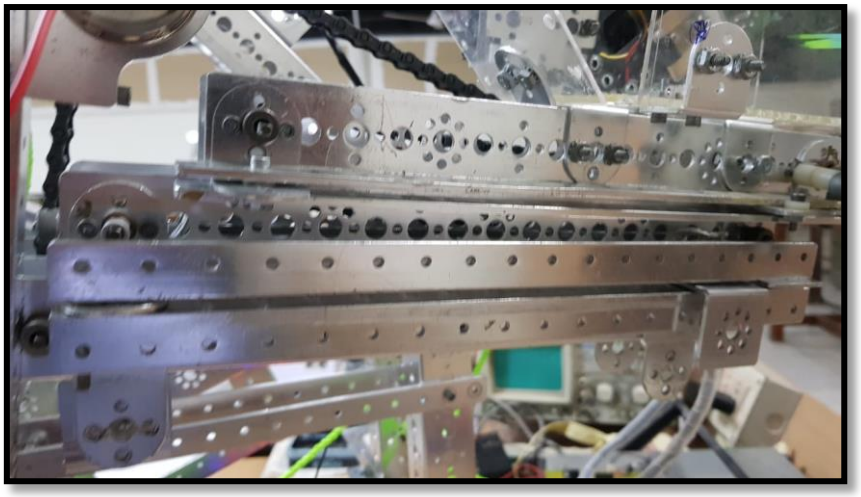
- [1]The rotation of the crank drives the linear movement the slider, or the expansion of gases against a sliding piston in a cylinder can drive the rotation of the crank.

- In the end game, the L clamp is manually attached and then beam is lifted using pulley mechanism.



4.2.3 Extending Arm:

- Tandem mechanism is used.



- The arm is both extendable and rotatable.
- It can make the collection box to move into the carter and rotate clockwise to land the minerals inside lander.
- This uses a TANDEM mechanism and driven by a tetrix motor.
- Double stage slider that can actuate using tandem chains can extend up to 1.5 Ft.
- As retracted it can accommodate inside the bot and when required it is extended.
- Bars that slide up and down on each other with rollers that support carriage on both sides
- Usually driven by a chain that runs up and down each bar on the lift.
- Because each bar is supported on both sides it can support heavy loads

- If not driven by one continuous chain, recommended that it is still driven both up and down to avoid jams

Sliding VS Telescoping Lift

➢ Telescoping Lift

• Does not support heavy loads efficiently

• Total size is only the size of the largest tube, adding stages does not directly mean making it larger

• Carriage must be in fixed position on last stage of lift

➢ Sliding Lift

• Can support heavy loads without pinching or binding

• Takes up extra space in robot for each stage

• Carriage can slide up last stage on lift, may be independently if needed

So we chose sliding lift.

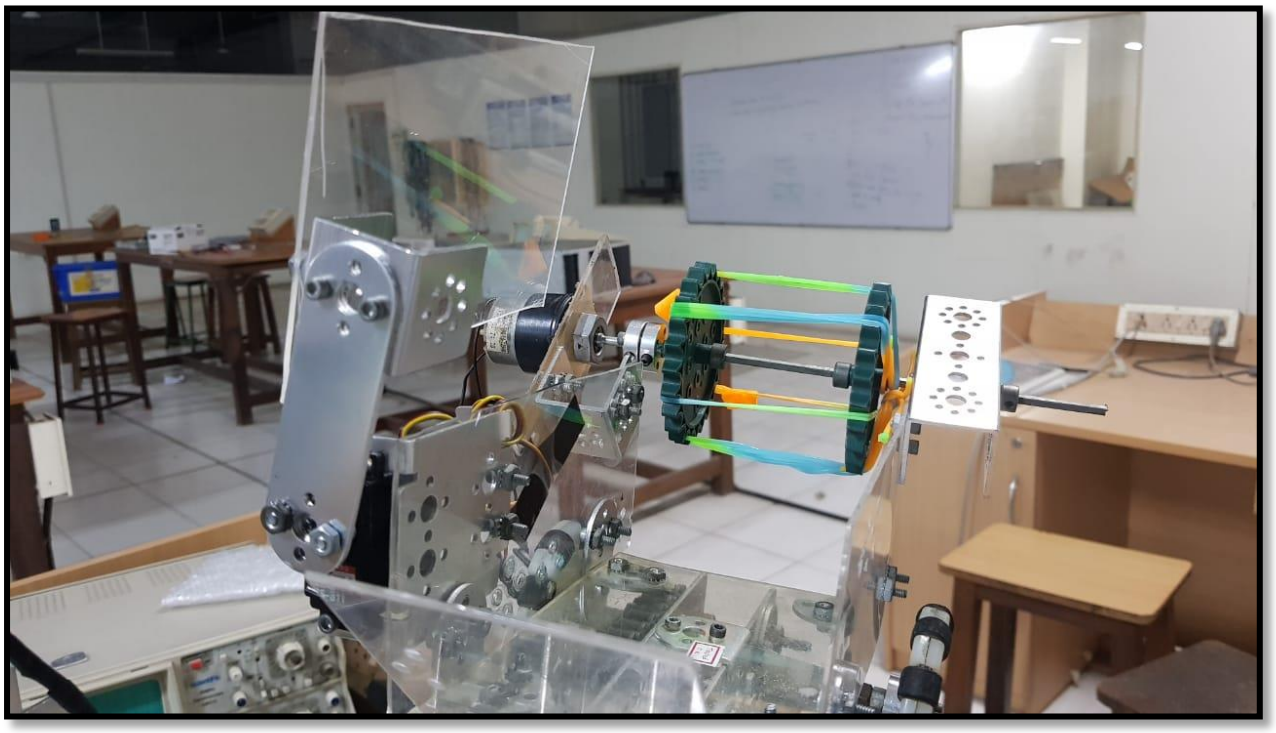
4.2.4 Collection box:

A low torque, high rpm motor drives a collector to collect balls and cubes in a sequence. The collection box has a capacity of 2 cubes or 2 balls at once.

- Gripper arm
- Rotary collector
- Wide collector

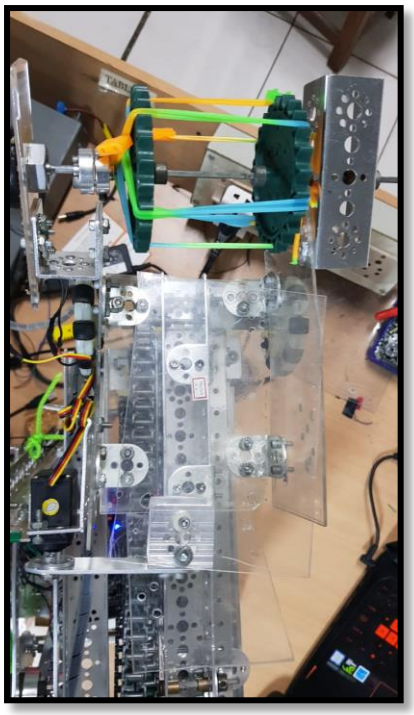
Gripper arm can collect only one at a time whereas, required collection rate of 2 minerals at a time makes ease of use of rotary collector.

This rotary collector can collect as well as drop the minerals if it was collected wrongly.



Some new ideas surrounding the Collection mechanism's prototypes were created. The prototypes will be developed to ensure it works more reliable and is less heavy.

This will spare us lots of time during the competitions. We made a list of the necessary materials, a few of them which we probably need to buy.



Using a servo door the dropping of the minerals into the lander is performed.

4.2.5 Mineral collection box movement:

1. The box is moved with a beam attached to a compound gear system to make a high torque.

2. It can roll from front side of the bot to back side.
3. Collecting box can collect or drop the minerals according to the position.
4. At collection it is fully descended, then the collection box collects and then beam is rotated up.
5. The collection box is aligned to lander and servo door is opened in order to drop minerals.
6. This increases the robot's maximum scoring potential while also allowing for new innovative ways of gameplay.

- There are 2 different rotary arms in bot.

- **Tandem arm**

- **Hanging mechanism**

Tasks:	Reflections
• **Tread-traction**	What did you learn from the stuff you did? • The main goal for today was to put the treads onto our existing drive train • Before we wheeled up and fitted the treads to the robot, we had to add rubber traction enhancing pieces to our treads for the big climb Reflection • The wheeling up was easy, as we already had a drive train, we just needed to add some new, tread-compatible wheels • Putting the rubber pieces on the treads was a lot

	harder than we anticipated: we learned that things that may seems easy might actually be difficult • The treads we added were satisfactory, but we did not think they would be ideal for climbing
• **Tread test**	• The moment of truth, the moment of learning • The treads did terribly. Worse than we anticipated • We learned that traction is a tricky topic to estimate, because on the robot traction acts VERY different from just basic tests not on the robot
• **Electronics strengthening**	• The electronics board was weak last session, so we added some nuts and bolts to ensure a safe and high quality wiring system.

4.3 Stability:

- As known bot looks taller, but the Centre of mass is balanced in between the 2 drive trains and the top.
- When arm is extended then center of mass moves to front of the bot.
- The battery placement is such that the Centre of mass is within the bot.
- The chance of toppling is less as CM is perfectly and it was practically tested.
- The bot can perform movements as well as rotate the collection box at a time.
- This helps bot for a faster performance.

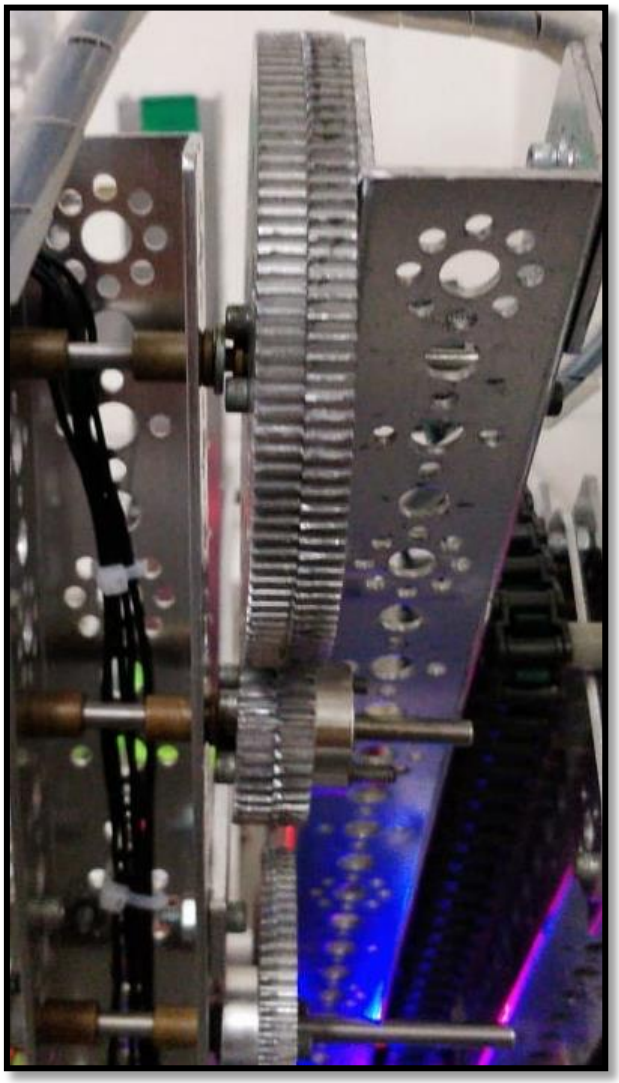
4.4 Mechanical parts Depends strategy:

Control/Possession Limits of Minerals – A Robot may Control or Possess a maximum of two Minerals at a time, however there is no Control or Possession limit on Minerals that are currently In the Crater.

- Ploughing through any quantity of Minerals is allowed but herding or directing multiple Minerals beyond the allotted maximum to gain a strategic advantage (i.e., Scoring, accessibility, defence) is not allowed.

4.5 Mechanical brainstorming:

- Usages of gears are so useful to increase torque.
- Gears tend to slip in-case of high power transmission.
- So we use double gear where it can transmit more power without any slip.



All tetrix motors have flattened shaft and the motor coupler can hold them with a grip screw. No slippage between the motor and coupler is observed.

But the Johnson motor has a M2 screw but the coupler has a m3 hole.

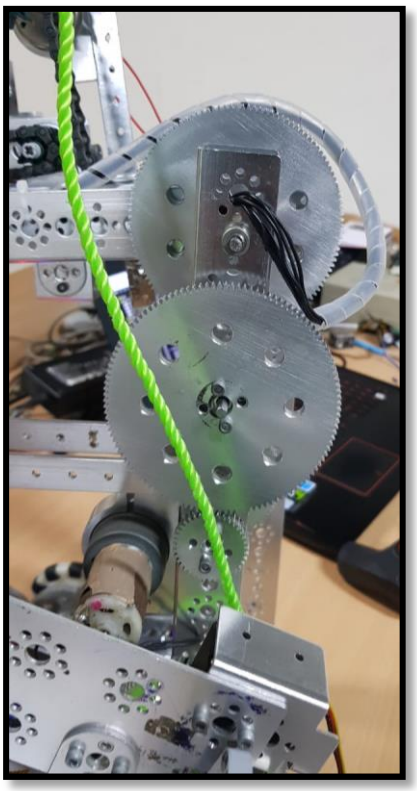
As the motor shaft is hardened, drilling is very difficult. Then a bench drill was used to make a larger hole in motor shaft.

M3 nut bolt was fit, and minimum ply of 7 degrees were observed.

Tandem mechanisms have a high friction between each stage in horizontal position

4.6 Speed:

- Our bot is planned to execute the strategy within the given time.
- As motors have both speed and torque without a compromise, this will be attained. (30 to 120) compound (30 to 120) compound (30 to 120)



4.6.1 Calculations:

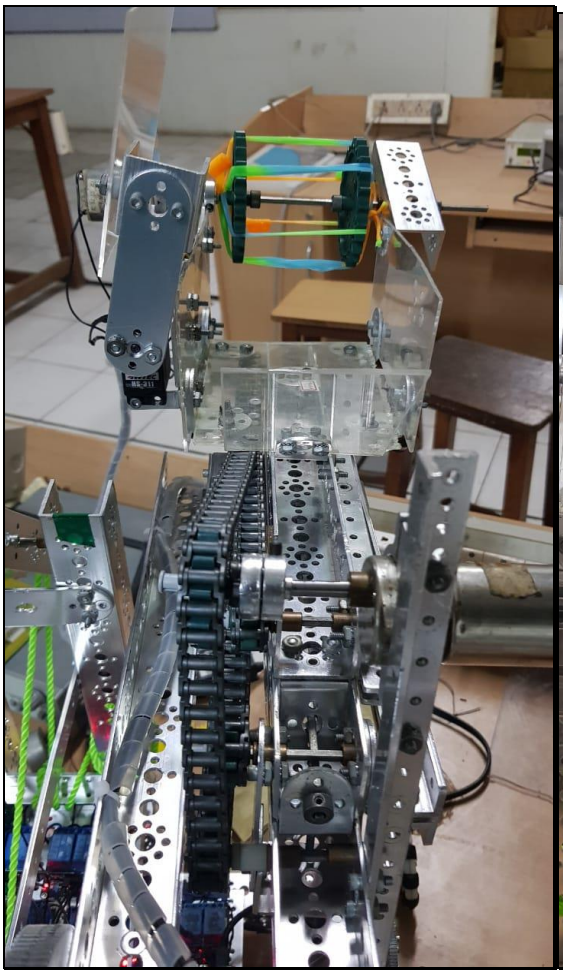
Input rpm = 150; then (1/3) times reduced. In compound (1/3) ^3 times. Torque is increased by 9 times; rpm reduced by 1/9 times which are more than sufficient.

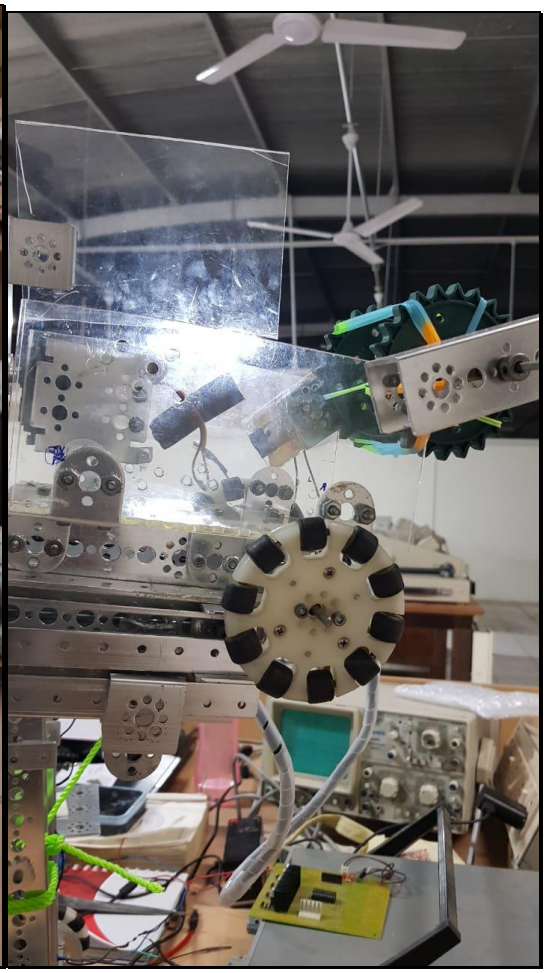
Output rpm = (150/9) = 16.6

Whereas torque is 9 times more and both are balanced.

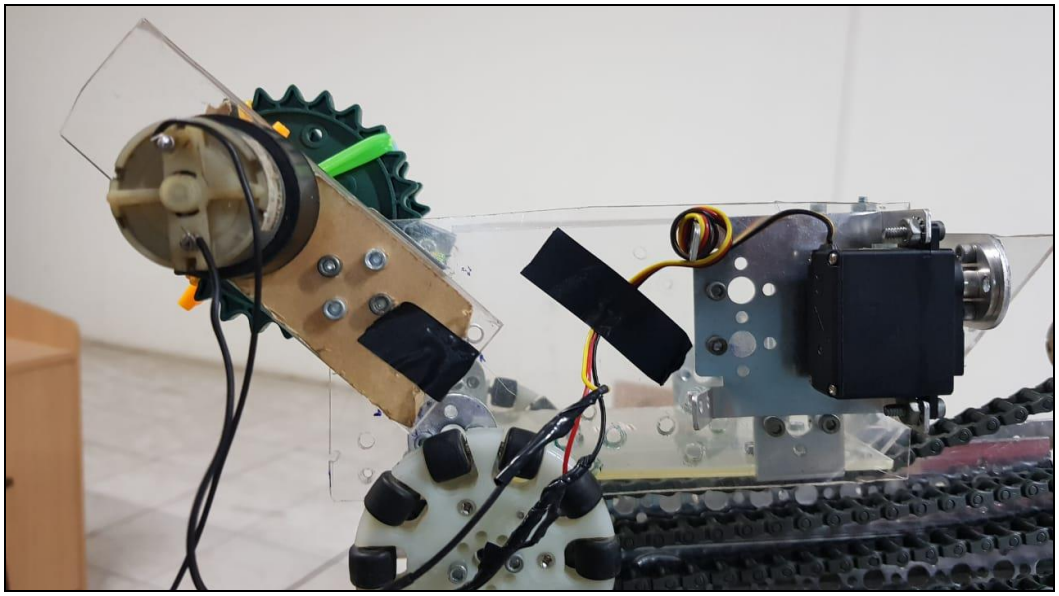
- In both autonomous and driver controlled period the bot is as fast as possible.
- Gear step down ratios include a high torque.
- Utilization of motors is planned as such that all motors will not draw high current at a time, but sequentially,
  which will not compromise the speed.

Collection box with servo door to drop minerals:





Collection box side view:



APPENDIX A : IMPLEMENTATION

```
import Adafruit_BBIO.PWM as PWM

import Adafruit_BBIO.GPIO as GPIO

import time

import evdev

from evdev

import InputDevice, categorize, ecodes

#Code to Input the Joystick

device = evdev.InputDevice('/dev/input/event1')

#Locomotion Motor Setup

GPIO.setup("P9_12", GPIO.OUT)

GPIO.setup("P9_15", GPIO.OUT)

GPIO.setup("P9_23", GPIO.OUT)

GPIO.setup("P9_27", GPIO.OUT)

#Locomotion Motor Initialization

GPIO.output("P9_12", GPIO.LOW)

GPIO.output("P9_15", GPIO.LOW)

GPIO.output("P9_23", GPIO.LOW)

GPIO.output("P9_27", GPIO.LOW)

#Latch Motor Setup

GPIO.setup("P8_9", GPIO.OUT)

GPIO.setup("P8_10", GPIO.OUT)

#Latch Motor Initialization

GPIO.output("P8_9", GPIO.LOW)

GPIO.output("P8_10", GPIO.LOW)
```

```
#Collection Mechanism Setup

#Mineral Arm Setup

GPIO.setup("P8_7", GPIO.OUT)

GPIO.setup("P8_8", GPIO.OUT)

#Extender Setup

GPIO.setup("P8_15", GPIO.OUT)

GPIO.setup("P8_16", GPIO.OUT)

#Mineral Collector Setup

GPIO.setup("P8_17", GPIO.OUT)

GPIO.setup("P8_18", GPIO.OUT)


#Collection Mechanism Initialization


#Mineral Arm Initialization

GPIO.output("P8_7", GPIO.LOW)

GPIO.output("P8_8", GPIO.LOW)


#Extender Initialization

GPIO.output("P8_15", GPIO.LOW)

GPIO.output("P8_16", GPIO.LOW)


#Mineral Collector Initialization


GPIO.output("P8_17", GPIO.LOW)

GPIO.output("P8_18", GPIO.LOW)
```

```
GPIO.cleanup()

#Servo Motor Setup
servoPin1 = "P9_14"
servoPin2 = "P9_16"

PWM.cleanup()

#Servo Motor Initialization
PWM.start(servoPin1, 95, 60)
PWM.start(servoPin2, 0, 60)

##############################################################################
'''MOTOR FUNCTIONS'''
##############################################################################

def forward_Run(t=0):
    print('Running Forward')
    GPIO.output("P9_12", GPIO.HIGH)
    GPIO.output("P9_15", GPIO.LOW)
    GPIO.output("P9_23", GPIO.LOW)
    GPIO.output("P9_27", GPIO.HIGH)
    time.sleep(t)

def backward_Run(t=0):
    print('Running Backward')
    GPIO.output("P9_12", GPIO.LOW)
```

```
    GPIO.output("P9_15", GPIO.HIGH)
    GPIO.output("P9_23", GPIO.HIGH)
    GPIO.output("P9_27", GPIO.LOW)
    time.sleep(t)

def turn_Right_All_Wheels(t=0):
    print('Turning Left - 2')
    GPIO.output("P9_12", GPIO.HIGH)
    GPIO.output("P9_15", GPIO.LOW)
    GPIO.output("P9_23", GPIO.HIGH)
    GPIO.output("P9_27", GPIO.LOW)
    time.sleep(t)

def turn_Right_Single_Wheel(t=0):
    print ('Turning Right - 1')
    GPIO.output("P9_12", GPIO.LOW)
    GPIO.output("P9_15", GPIO.LOW)
    GPIO.output("P9_23", GPIO.LOW)
    GPIO.output("P9_27", GPIO.HIGH)

   time.sleep(t)

def turn_Left_All_Wheels(t=0):

    print('Turning Right - 2')
```

```
 GPIO.output("P9_12", GPIO.LOW)

   GPIO.output("P9_15", GPIO.HIGH)

GPIO.output("P9_23", GPIO.LOW)
 GPIO.output("P9_27", GPIO.HIGH)
  time.sleep(t)

def turn_Left_Single_Wheel(t=0):
 print ('Turning Left -1')
 GPIO.output("P9_12", GPIO.HIGH)
GPIO.output("P9_15", GPIO.LOW)
 GPIO.output("P9_23", GPIO.LOW)
 GPIO.output("P9_27", GPIO.LOW)
time.sleep(t)

def latch_Up(t=0):
 print('Latch Rising UP')
 GPIO.output("P8_9", GPIO.HIGH)
 GPIO.output("P8_10", GPIO.LOW)
time.sleep(t)

def latch_Down(t=0):
print('Latch Rising DOWN')
 GPIO.output("P8_9", GPIO.LOW)
```

```
 GPIO.output("P8_10", GPIO.HIGH)
time.sleep(t)


def mineral_Arm_Up(t=0):
   print('Mineral Arm Rising')
 GPIO.output("P8_7", GPIO.LOW)
 GPIO.output("P8_8", GPIO.HIGH)
 time.sleep(t)

def mineral_Arm_Down(t=0):
  print('Mineral Arm Lowering')
  GPIO.output("P8_7", GPIO.HIGH)
 GPIO.output("P8_8", GPIO.LOW)
 time.sleep(t)

def extender_Out(t=0):
  print('Mineral Collector Extending')
 GPIO.output("P8_15", GPIO.HIGH)
 GPIO.output("P8_16", GPIO.LOW)
  time.sleep(t)

def extender_In(t=0):
   print('Mineral Collector Retracting')
  GPIO.output("P8_15", GPIO.LOW)
```

```
GPIO.output("P8_16", GPIO.HIGH)
 time.sleep(t)

def mineral_Collector_In(t=0):
print('Mineral In')
GPIO.output("P8_17", GPIO.HIGH)
GPIO.output("P8_18", GPIO.LOW)
time.sleep(t)

def mineral_Collector_Out(t=0):
  print('Mineral Out')
  GPIO.output("P8_17", GPIO.LOW)
 GPIO.output("P8_18", GPIO.HIGH)
 time.sleep(t)

def motor_Stop(t=0):
   print('Motors are Stopped')
   GPIO.output("P9_12", GPIO.LOW)
  GPIO.output("P9_15", GPIO.LOW)
 GPIO.output("P9_23", GPIO.LOW)
GPIO.output("P9_27", GPIO.LOW)
 GPIO.output("P8_9", GPIO.LOW)
 GPIO.output("P8_10", GPIO.LOW)
 GPIO.output("P8_8", GPIO.LOW)
```

```
GPIO.output("P8_8", GPIO.LOW)

 GPIO.output("P8_16", GPIO.LOW)

 GPIO.output("P8_16", GPIO.LOW)

GPIO.output("P8_17", GPIO.LOW)

GPIO.output("P8_18", GPIO.LOW)

time.sleep(t)



################################################################################
'''SERVO FUNCTIONS'''
################################################################################


def latch_Servo(servoPin, desiredAngle):

  dutyCycle=11./120.*desiredAngle + 6

 PWM.set_duty_cycle(servoPin, dutyCycle)


def marker_Servo(servoPin, desiredAngle):

 dutyCycle=11./120.*desiredAngle + 6

PWM.set_duty_cycle(servoPin, dutyCycle)


################################################################################
#Autonomous Functions
################################################################################


def image_Process(ret):

 try:

    camera = VideoCapture(0)
```

```
 ret, frame = camera.read()

    blurValue = 41

blur = GaussianBlur(frame, ((blurValue, blurValue)), 0)
 hsv = cvtColor(blur, COLOR_BGR2HSV)
 lower = [18,50,50]
 upper = [35,255,255]
lower = array(lower, dtype = "float32")
upper = array(upper, dtype = "float32")
mask = inRange(hsv, lower, upper)

 output = bitwise_and(frame, hsv, mask = mask)
 imwrite("original5.jpg", frame)
_,contours, hierarchy = findContours(mask, RETR_TREE, CHAIN_APPROX_SIMPLE)

for cnt in contours:
 approx = approxPolyDP(cnt, 0.01*arcLength(cnt, True), True)
 drawContours(mask, [approx], 0, (0), 5)
 x = approx.ravel()[0]
 y = approx.ravel()[1]

 if len(approx)<10 :
 approx = approxPolyDP(cnt, 0.01*arcLength(cnt, True), True)
 drawContours(output, [approx], 0, (100), 10)
```

```
 print(x)
else:
 drawContours(output, [approx], 0, (100), 10)
       imwrite("Shapes5.jpg", output)
      break

 if x>=0 and x<=213 :
return -1
   elif x>213 and x<=427:
      return 0
 elif x>427 and x<=640:
     return 1


  except Exception as e:
    print(e)
 print('\n Process abrupted!!')
    pass

def autonomous_One():
 print('Autonomous Strategy 1 Running\n\n')
 '''
latch_Down(2.0)
   print('Bot Lowered')
   motor_Stop()
```

```
   backward_Run(0.5)
   print('Bot Grounded')
   motor_Stop()
 time.sleep(0.5)
  latch_Servo("P9_14", 90)
   print('Servo Unlocked')
   time.sleep(0.5)

   #marker servo test
  marker_Servo("P9_16",90)
   print('Marker_Released')
   motor_Stop()
   time.sleep(2)
  marker_Servo("P9_16",0)
 print('Servo_closed')
   motor_Stop()

   '''
 direction = image_Process(0)

   if (direction == 0):
  print('Gold is on the Centre')
##Timing for the battery voltage 12.71V
      forward_Run(2.2)
```

```
        motor_Stop()
        time.sleep(0.2)
    turn_Right_All_Wheels(0.4)
    motor_Stop()
     time.sleep(0.2)
     marker_Servo("P9_16",90)
      print('Marker_Released')
       motor_Stop()
      time.sleep(2)
       marker_Servo("P9_16",0)
     print('Servo_closed')
       motor_Stop()
       backward_Run(0.5)
       motor_Stop()
      turn_Left_All_Wheels(1)
       motor_Stop()
     time.sleep(0.2)
       forward_Run(3.2)
       motor_Stop()


    elif (direction == 1):
      print('Gold is on the right')
     forward_Run(0.2)
```

```
motor_Stop()
 time.sleep(0.5)
 turn_Left_All_Wheels(0.2)
 motor_Stop()
 time.sleep(0.3)
 forward_Run(1.6)
 motor_Stop()
time.sleep(0.2)
turn_Right_All_Wheels(0.6)
 motor_Stop()
time.sleep(0.2)
 forward_Run(1)
motor_Stop()
 marker_Servo("P9_16",90)
 print('Marker_Released')
 time.sleep(1.5)

marker_Servo("P9_16",0)

 print('Servo_closed')
time.sleep(0.2)
 backward_Run(0.5)
motor_Stop()
```

```
        turn_Left_All_Wheels(1)
         motor_Stop()
         time.sleep(0.2)
        forward_Run(3.2)
         motor_Stop()

    elif (direction == -1):
         print('Gold is on the Left')
         forward_Run(0.2)
       motor_Stop()
         time.sleep(0.5)
         turn_Right_All_Wheels(0.25)
       motor_Stop()
         time.sleep(0.3)
         forward_Run(1.4)
       motor_Stop()
         time.sleep(0.2)
      turn_Left_All_Wheels(0.55)
         motor_Stop()
       time.sleep(0.2)
         forward_Run(1.05)
         motor_Stop()
        time.sleep(0.2)
```

```
     turn_Right_All_Wheels(0.5)
       motor_Stop()
    time.sleep(0.2)
      marker_Servo("P9_16",90)
    print('Marker_Released')
       time.sleep(1.5)
       marker_Servo("P9_16",0)
      print('Servo_closed')
 time.sleep(0.2)
#backward_Run(0.5)
 #motor_Stop()
#turn_Left_All_Wheels(1)
     #motor_Stop()
 #time.sleep(0.2)
 #forward_Run(3.2)
  motor_Stop()


   motor_Stop()
 print('Autonomous complete!!!')
################################################################################
################################################################################
'''MAIN FUNCTION'''
################################################################################

print('All Initialisations Complete, Bot is Ready\nRock the Stage')
```

```
try:

    for event in device.read_loop():
 if event.type == ecodes.EV_KEY:
       k = categorize(event)
    #print (k)

   if (k.keycode == 'BTN_START'):
            if (k.keystate == 1):
               autonomous_One()

              elif (k.keystate == 0):
  pass
           elif (k.keycode == 'BTN_SELECT'):
             if (k.keystate == 1):
    GPIO.cleanup()
                 PWM.cleanup()
              break

 elif (k.keystate == 0):
       continue

elif (k.keycode == 'BTN_TR'):
          if (k.keystate == 1):
             turn_Right_Single_Wheel()
```

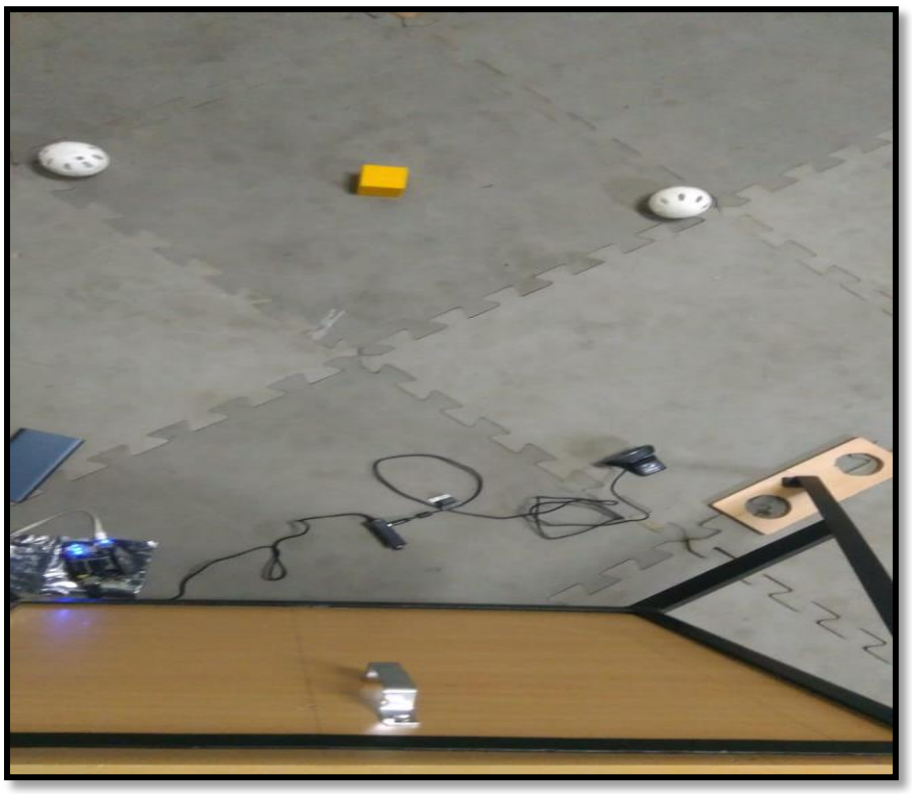
```
            elif (k.keystate == 0):
                motor_Stop()
          elif (k.keycode == 'BTN_TL'):
            if (k.keystate == 1):
      turn_Left_Single_Wheel()
             elif (k.keystate == 0):
   motor_Stop()
    elif (k.keycode == 'BTN_TR2'):
   if (k.keystate == 1):
 mineral_Collector_In()
  elif (k.keystate == 0):
   motor_Stop()
   elif (k.keycode == 'BTN_TL2'):
   if (k.keystate == 1):
mineral_Out()
    elif (k.keystate == 0):
     motor_Stop()
   elif (k.keycode == ['BTN_WEST', 'BTN_Y']):
       if (k.keystate == 1):
print('Servo Out')
latch_Servo("P9_14", 90)
      elif (k.keystate == 0):
pass
  elif (k.keycode == ['BTN_A', 'BTN_GAMEPAD', 'BTN_SOUTH']):
       if (k.keystate == 1):
print('Servo In')
latch_Servo("P9_147", 0)
```

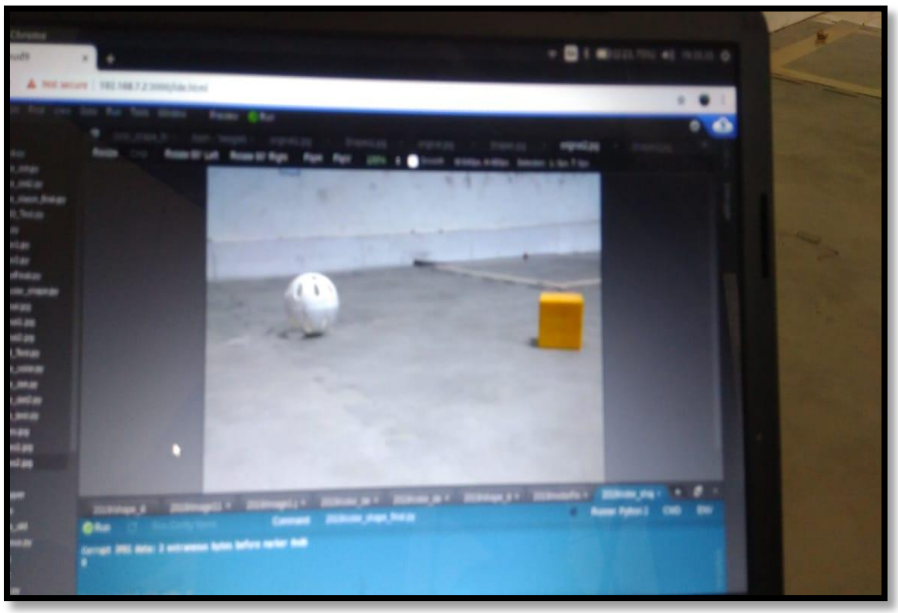
```
 elif (k.keystate == 0):


pass

 elif (k.keycode == 'BTN_THUMBR'):

if (k.keystate == 1):

print('Importing Required Packages to start Image Processing')

from cv2 import VideoCapture, GaussianBlur, COLOR_BGR2HSV, cvtColor, bitwise_and, inRange,
imwrite, findContours, approxPolyDP, drawContours, arcLength, CHAIN_APPROX_SIMPLE,
RETR_TREE

 from numpy import array

    print('Import Completed')

            elif (K.keystate == 0):

             pass

   elif event.type == ecodes.EV_ABS:

        k = event

#print (k)

 if (k.code == 17):

  if (k.type == 03):

    if (k.value == -1):

         mineral_Arm_Up()

              elif (k.value == 00):

                motor_Stop()

            elif (k.value == 01):

  mineral_Arm_Down()

elif (k.code == 16):

if (k.type == 03):

  if (k.value == -1):

    extender_In()
```

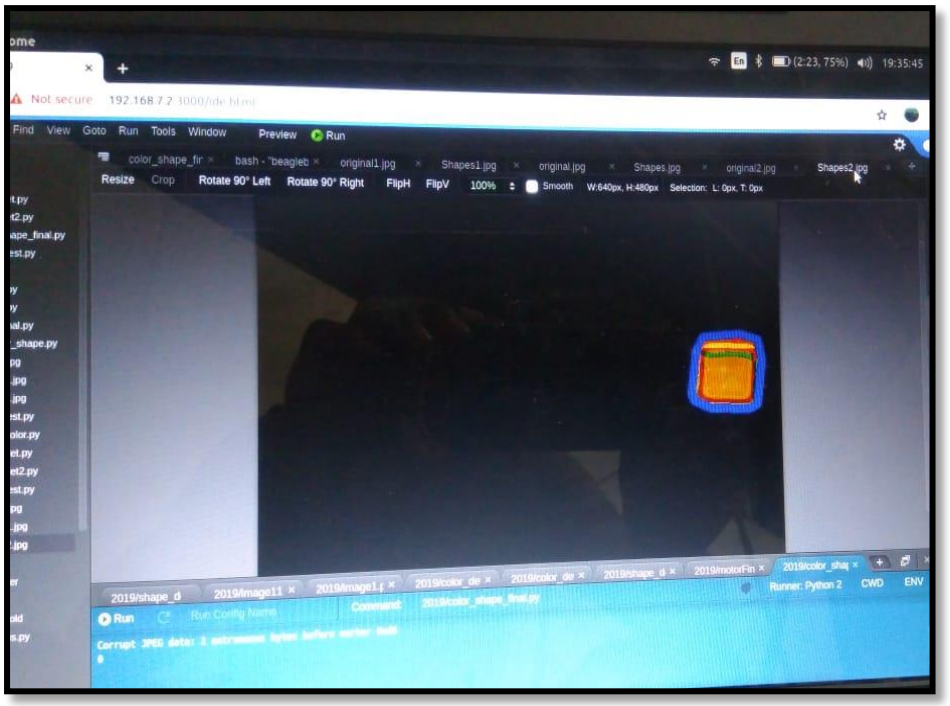
```
        elif (k.value == 00):
  motor_Stop()
  elif (k.value == 01):
extender_Out()
  elif (k.code == 01):
 if (k.type == 03):
  if (k.value >= 00 and k.value <=108):
                forward_Run()
              elif (k.value >= 148 and k.value <= 255)
                 backward_Run()
       else:
                    motor_Stop()
     elif (k.code == 00):
            if (k.type == 03):
         if (k.value >= 148 and k.value <=255):
                  turn_Left_All_Wheels()
    elif (k.value >= 00 and k.value <= 108):
                   turn_Right_All_Wheels()
         else:
  motor_Stop()
           elif (k.code == 05):
     if (k.type == 03):
      if (k.value >= 00 and k.value <= 108):
             latch_Up()
       elif (k.value >= 148 and k.value <= 255):
         latch_Down()
  else:
```

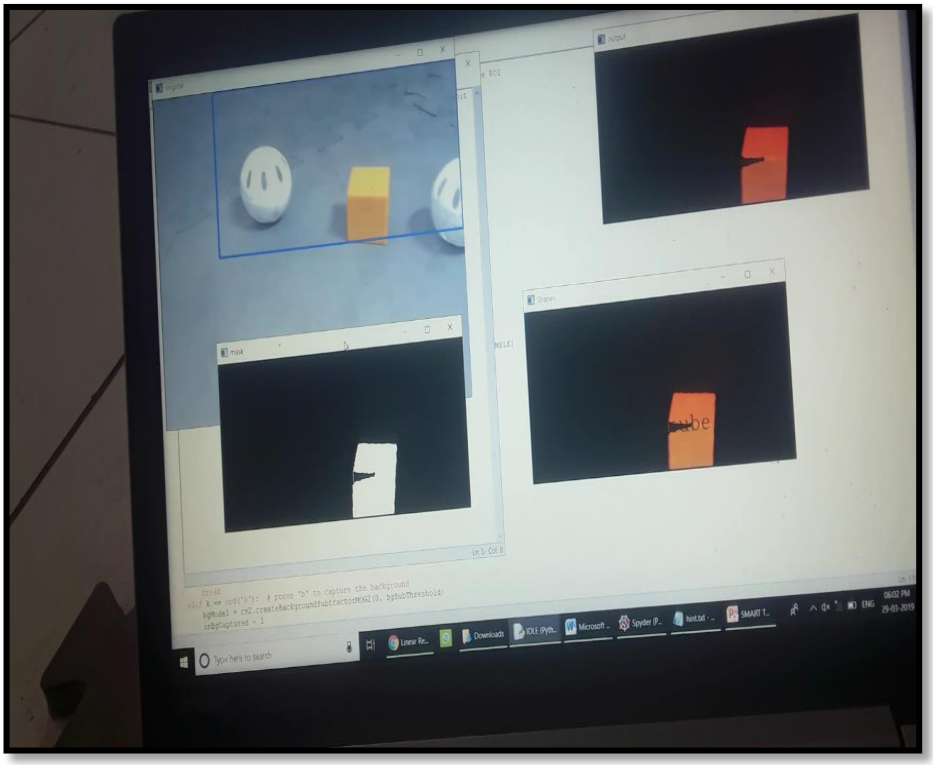
```
    motor_Stop()
          elif (k.code == 02):
       if (k.type == 03):
     if (k.value >= 00 and k.value <= 108):
            pass
elif (k.value >= 148 and k.value <= 255)
           pass
  else:
pass
except Exception as R:
print(R)
motor_Stop()
 GPIO.cleanup()
   PWM.cleanup()
finally:
motor_Stop()
GPIO.cleanup()
PWM.cleanup()
print('Thank You, Have a Good Day')
```

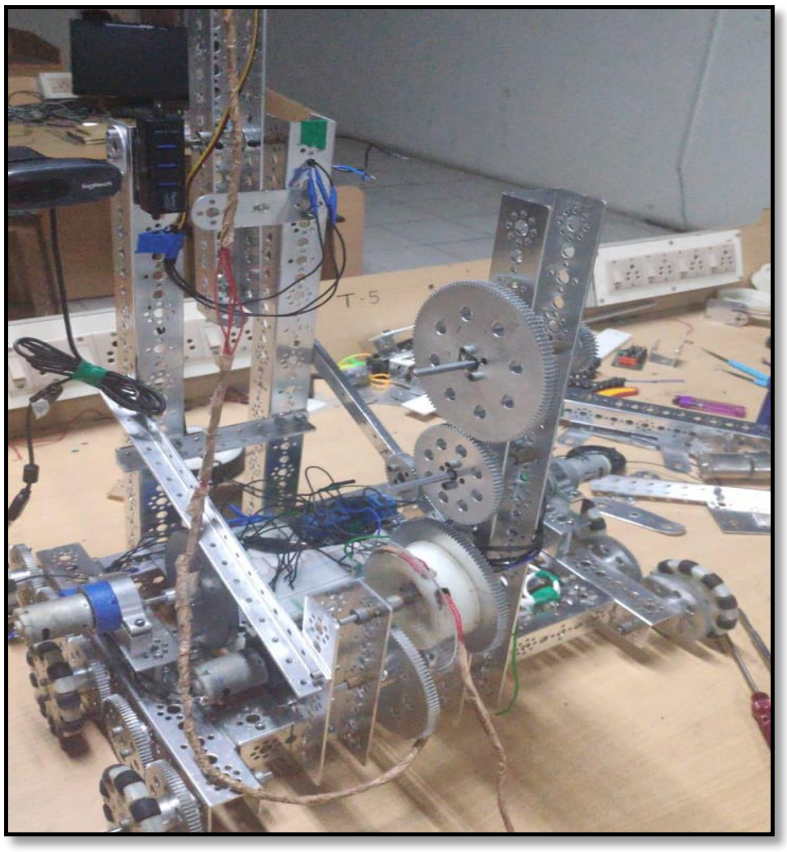
APPENDIX B -OUTPUT IMAGES






Not secure
Find View Goto Run Tools Window Preview Run
Resize Crop Rotate 90° Left Rotate 90° Right FlipH FlipV 100%
Run
Run Config Name




T-5

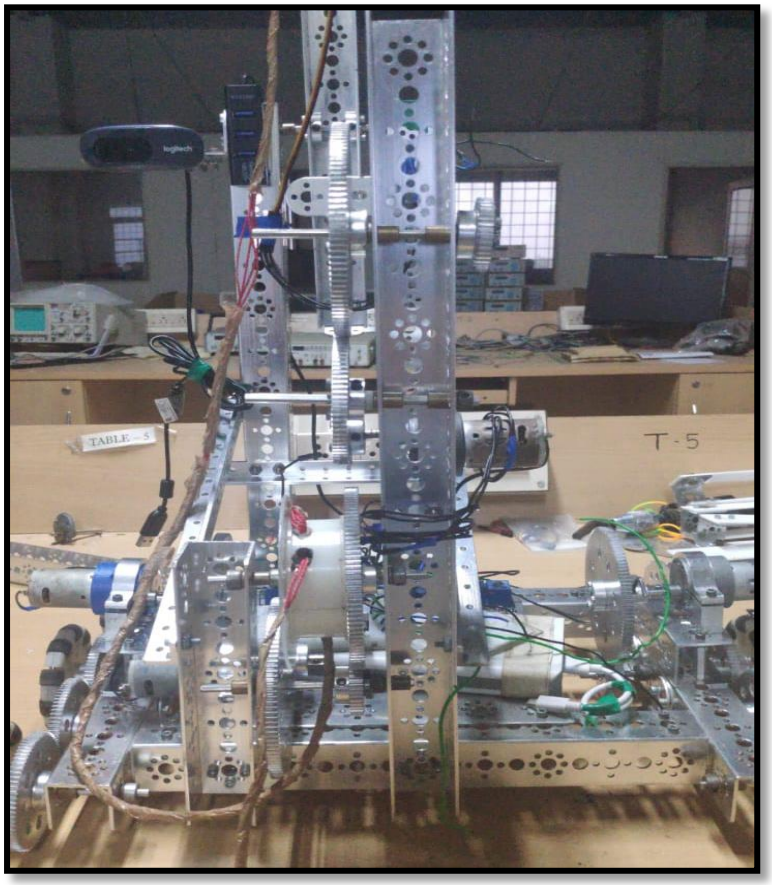


TABLE - 5
T-5