Drone-Assisted Energy Delivery

Team 40

Team members & roles:

- **Automation Specialists:**

  Drew Underwood

  Garrett Lies

  Khalifa Al Dhaheri.

- **Image processing:**

  Abdullah Al Oabaidi

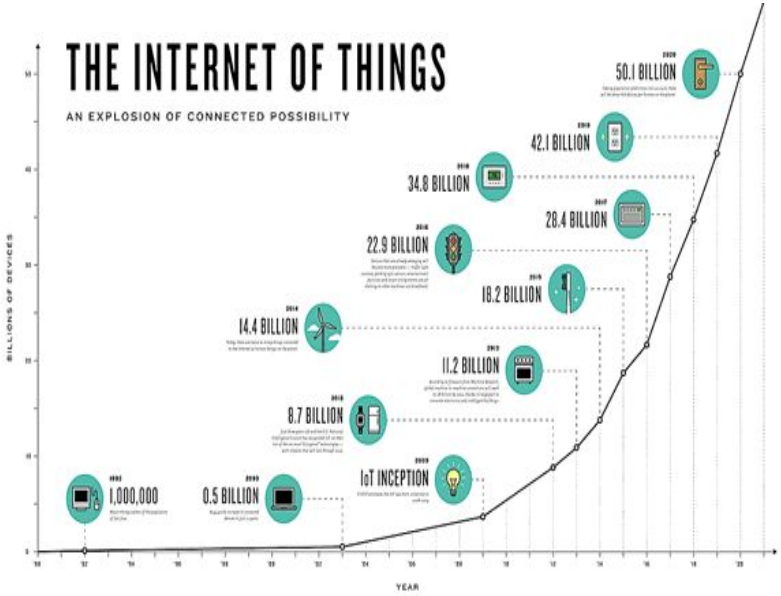
- **Landing/Charging Station:**

  Ahmed Al Hulayel.

Advisors: Randall Geiger & Degang Chen

Problem Statement

- The imminent demand of the IOT (Internet of Things) Boom
- Not all IOT Nodes can be expected to reliably connect to the power grid
- Cost to maintain connection to all nodes is expensive
- Use drones to deliver energy autonomously



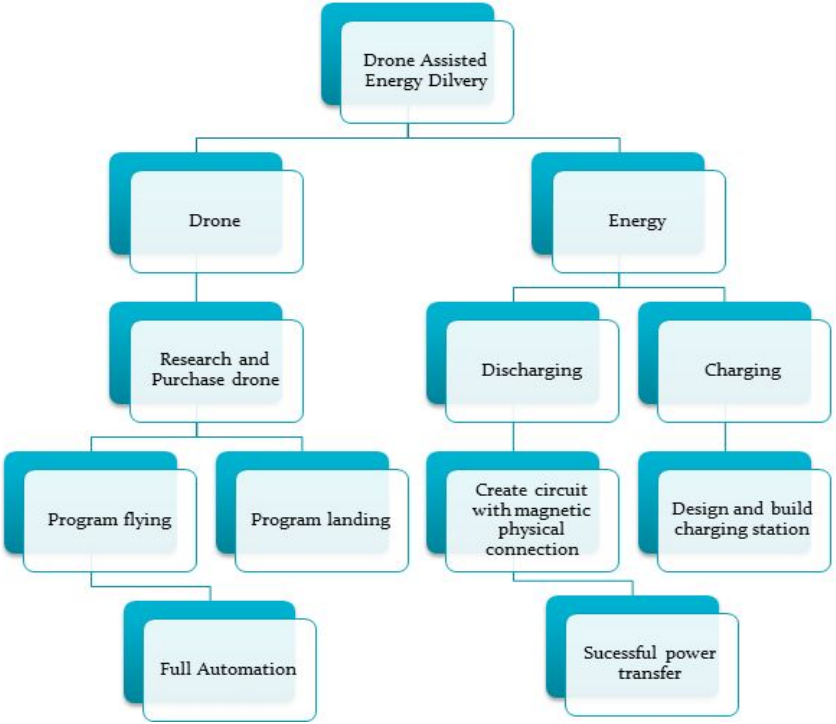
Requirements

For this project we have been tasked with

- A fully autonomous solution for delivering energy by drone from one location to another, the drone should take off, fly to the location, land, charge the device, and return home
- A landing station to allow the drone to connect to the device being charged

Project Plan

- Two teams: Automation and Landing Station
- Research Drone
- Establishing a connection
- Landing Station
- Automation
- Image Processing



Drone Research

- Two choices - DJI and Intel Aero
- DJI: Least Expensive Option, included a 4K camera, and library support. Hardware is not flexible, updates depreciate code, must login to DJI servers with a private key to use
- Intel Aero Ready to Fly: Customizable hardware, several libraries. Small community, lower level, no dedicated automation library
- Ended up going with Intel Aero for flexibility

Establishing a Connection

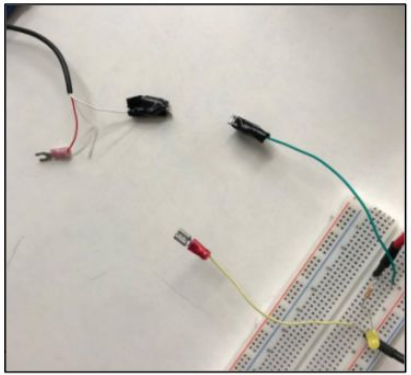
Wire Connections using magnets:

Establish a connection using conducting magnets - similar to MacBooks

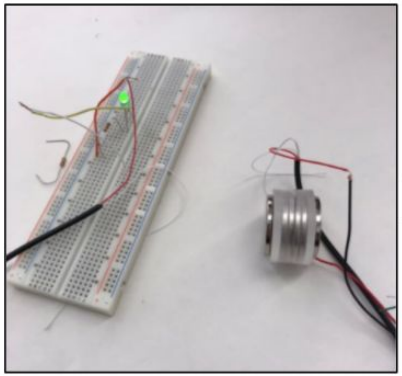
- Easier to design
- More efficient transfer
- Demands higher accuracy - harder to land

Old design

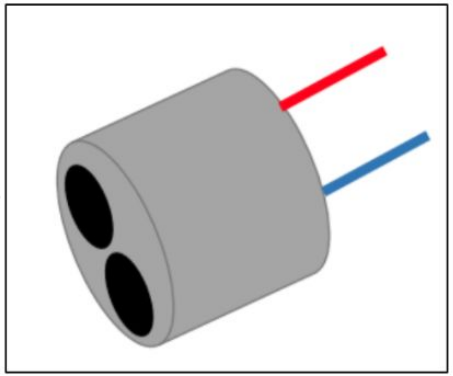
Connection with magnet I



Connection with magnet II

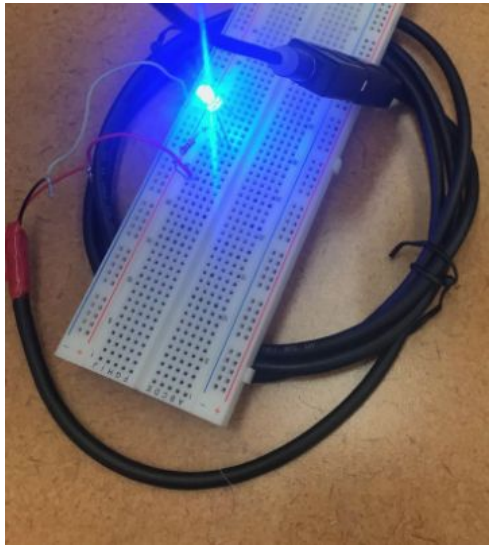


Latest design

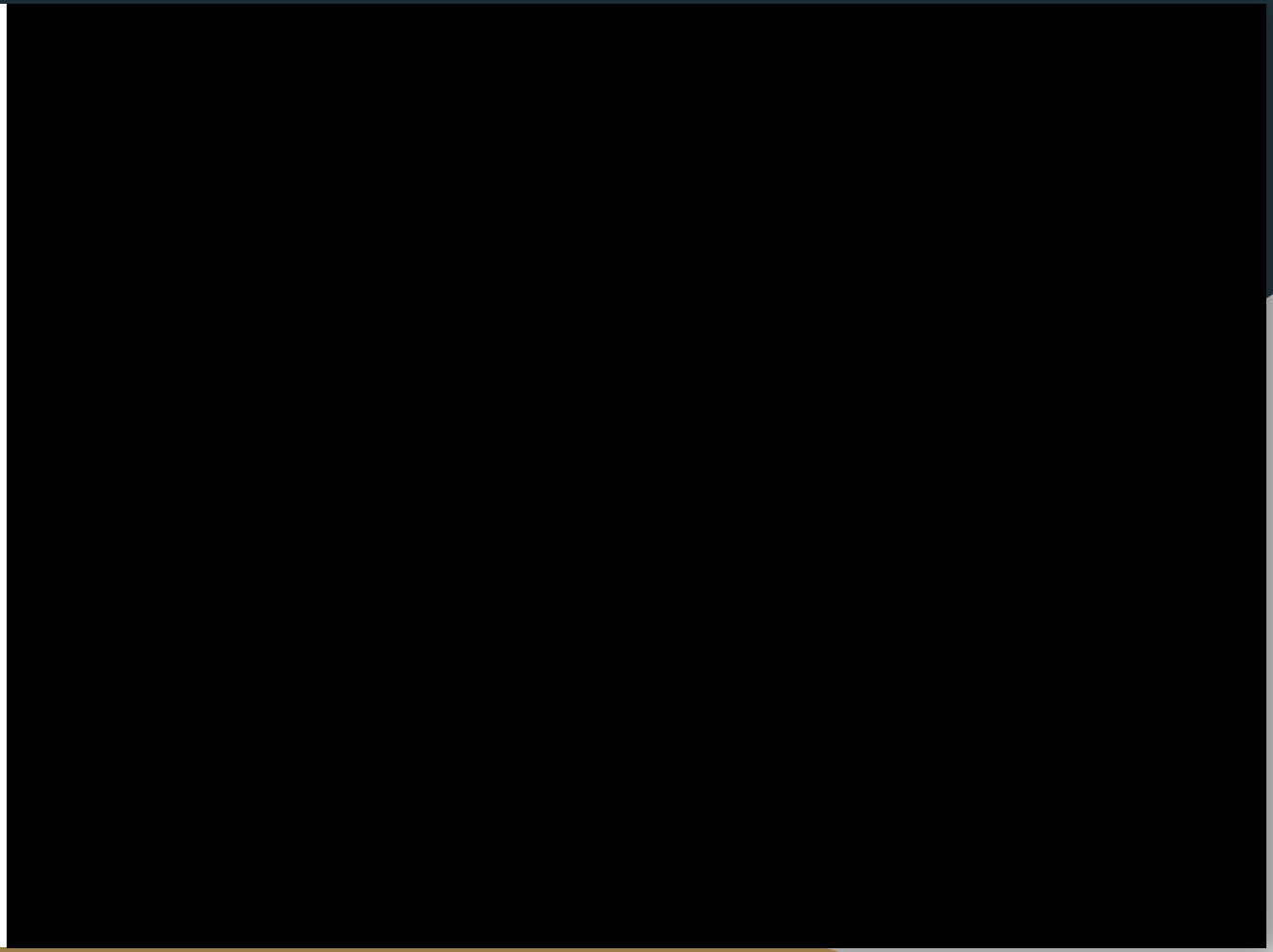


Most of testing done on LED for safety

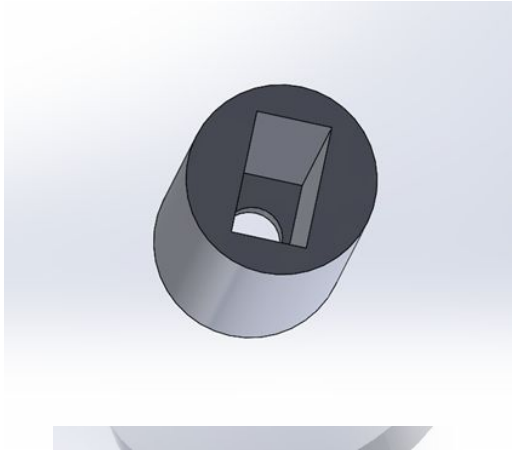
New design

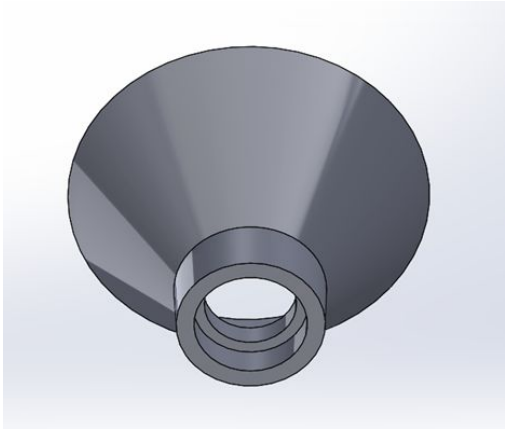


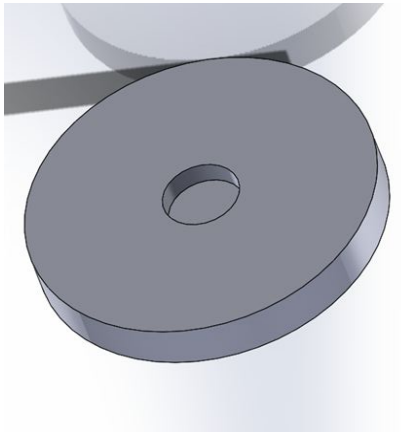
Demo #1



SolidWorks







Landing Station Old/New

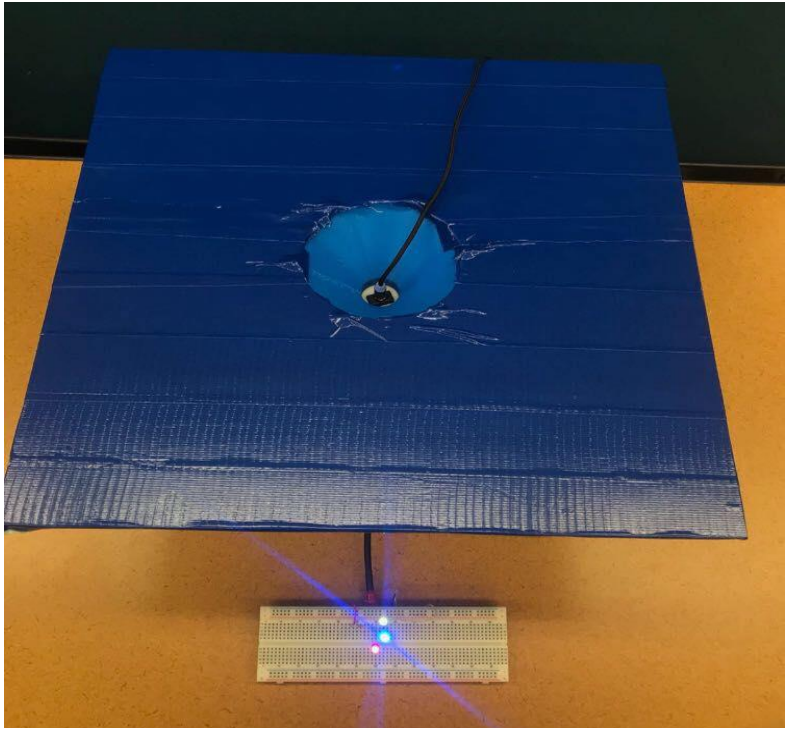
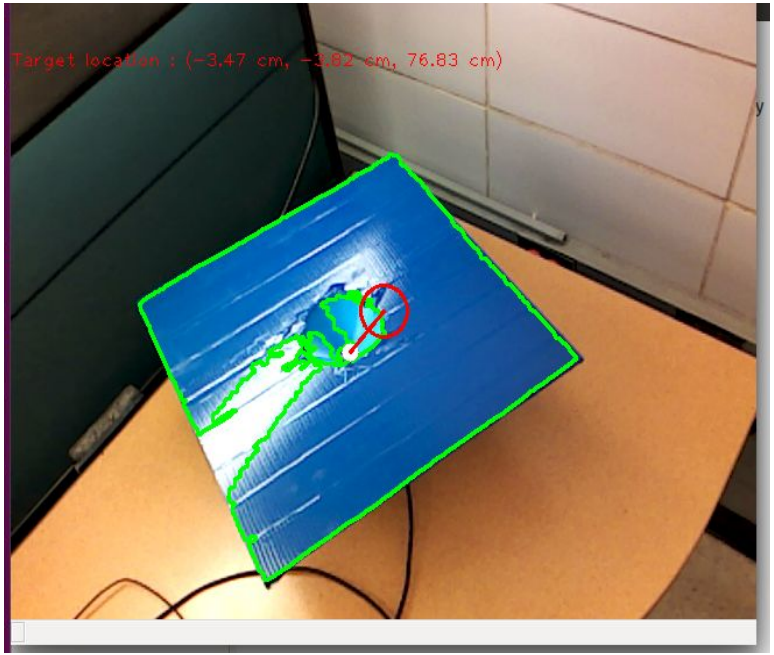


Image processing

- Done in Python
- Greatly dependent on 3rd party library

  OpenCv
- Filter Color and Size
- Find location in x,y,z coordinates



Functions Used:

1. Create Color Mask
2. Find the Centroid of the Object
    a. Filter for size
3. Using Triangular Similarity to find distances:
    a. Focal length: (P x D)/W
    b. Distance z: (W x F)/ P
    c. Distance x,y: (D x P)/F

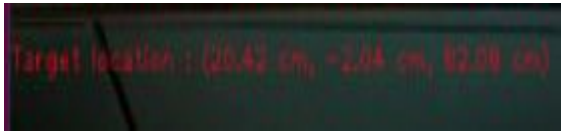
```
def create_mask(color, image) :
```

```
def find_centroid (mask, original, obj_width, focalLength):
```

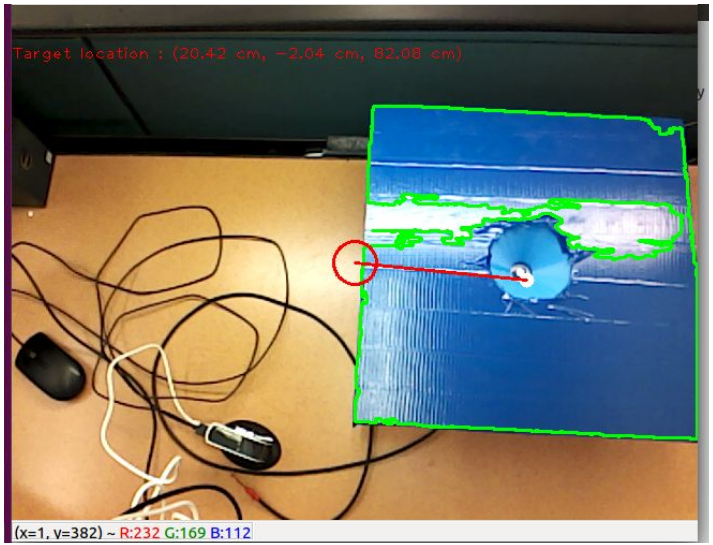
```
def find_distance(fixed_width, focalLength, perWidth ):
```

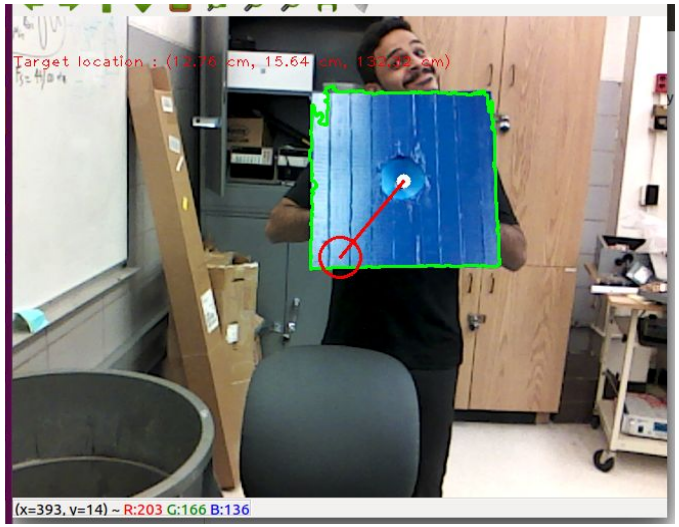
```
def find_pos_diff (cX, cY, dX, dY, focal_length, distance):
```

Outputs:



Filtering noise





Automation

- Compute Board communicates with flight controller by sending Mavlink messages
- Mavlink messages contain important information and commands for the drone to carry out

Overall implementation

- Dynamically create waypoints
- Take off and fly to general landing station location with GPS
- Use image processing to guide the drone towards the landing station
- Mimic movements towards landing station by creating and canceling waypoints
- Land Drone on station, charge the device, and return home

Dronekit

- High level library, easy to use and understand, documentation includes examples
- Includes the implementation of many Mavlink Messages including, take off and landing, GPS waypoints, velocity and position control, vehicle state, gimbal and rotation control, and many more

Testing

- Used an incremental approach to testing
- Started out with simple cases, arming and disarming, take off and landing, etc
- Wrote many tests to experiment with control
- Library restricted use of most commands without a GPS lock, had to test outside

Example Shown is a velocity test to see if we could control the speed and direction of the vehicle

```
# Connect to the Vehicle.
print "Connecting"
vehicle = connect('tcp:127.0.0.1:5760', wait_ready=False)
print "Connected"

# Change to Guided mode
print "Changing mode to GUIDED"
PX4setMode(8)
time.sleep(1)

#Arm motors
print "Arming motors:"
vehicle.armed = True
while not vehicle.armed:
    print "waiting for arming"
    time.sleep(1)

#Take off
#north, east, down, duration (in m/s)
#try flying upwards at 0.5m/s for 1 second
print "Taking Off"
send_ned_velocity(3,3,-3,5)

#Land
#try landing, fly downwards at 0.5m/s for 1 second
print "Landing"
send_ned_velocity(3,3,3,5)

vehicle.armed = False
# Close vehicle object before exiting script
vehicle.close()

print "Completed"
```

Results

Completed

- Arming and Disarming
- Take off and landing with GPS (stability issues)
- Movement between waypoints
- Entire project implemented in code, just need another month or two to fix stability issues

Take off Stability Issues

Sometimes tests don’t end so well...

Conclusion

- We were able to successfully implement the requirements for the landing station and image processing
- While we were not able to meet all requirements for the automation, we are confident we could complete the project with a little bit more time, especially as the weather clears up
- Implemented a final solution to the automation requirement, better solutions will be available as full support for Dronekit on PX4 becomes available

Questions?