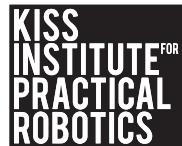


Oklahoma Drone workshop

Dr Amy McGovern

Dr Andrew Fagg

School of Computer Science and School of Meteorology
University of Oklahoma



Today's Schedule

- Morning combined session
 - Drone safety
 - Drone catching
 - Connecting to the drones
 - Ensuring we can all fly
- Morning session novice
 - Variables and objects
 - Conditionals
 - Loop basics
- Morning session advanced
 - Variables and objects
 - User input
 - Conditionals
 - Looping
 - Sensors: Vision

Turning: Two approaches

- Yaw-based turning

```
fly_direct(roll=0,pitch=0,yaw=#,vertical_movement=0,duration=#)
```

You can combine roll/pitch/yaw to fly curves

- Sensor-based turning

```
turn_degrees(degrees)
```

Sample code

```
from pyparrot.Minidrone import Mambo

# create the drone object and connect
drone = Mambo(use_wifi=True)
drone.connect(num_retries=3)

# maximum speeds - DO NOT CHANGE
drone.set_max_tilt(10)
drone.set_max_vertical_speed(1)

drone.flat_trim()

# takeoff
drone.safe_takeoff(5)

# fly forward a small amount for one second
drone.fly_direct(roll=50, pitch=10, yaw=0, vertical_movement=0, duration=1)

# land and disconnect
drone.safe_land(5)
drone.disconnect()
```

Your turn!

Explore parameters

Test:

roll , yaw, vertical_movement

from -100 to 100

Duration is in seconds (max of 2)

Explore turn_degrees

Multiple Takeoffs/Landings/Turning

- Goal: deliver blood to two hospitals
- Task analysis:
 - Fly to first hospital
 - Land
 - Take off after first landing
 - Turn
 - Fly to the 2nd hospital
 - Land at 2nd hospital
 - Disconnect

Your turn!

Get those drones delivering blood to BOTH hospitals in the same program! No touching your drone mid-way and no restarting your program after the first landing. Write ONE program to land at both hospitals.

Use what you just learned about turning and roll/pitch/yaw to move from the first hospital to the second



Drone art

Winter Olympics

<https://www.youtube.com/watch?v=fCd6P7Ya160>

Time Magazine

<https://www.youtube.com/watch?v=JGjmRRTThdk>

User Input: Part 1

- Programs are more interesting with input! We can use the input to change the behavior of our programs.
- Start a new python file in your editor and type

```
name = input("What is your name?")
```

- `input()` is a python function that prints the text you give it and waits for you to type a response AND hit return/enter

User Input: Part 2

- Now add the 2nd line to your program

```
name = input("What is your name?")
print("Nice to meet you %s" % name)
```

- Run the program. What does it do?
- `print()` prints everything inside the parentheses to the screen
- `"%s" % name`
 - Transforms the variable *name* into a string that can be printed

Conditionals Part 1

If

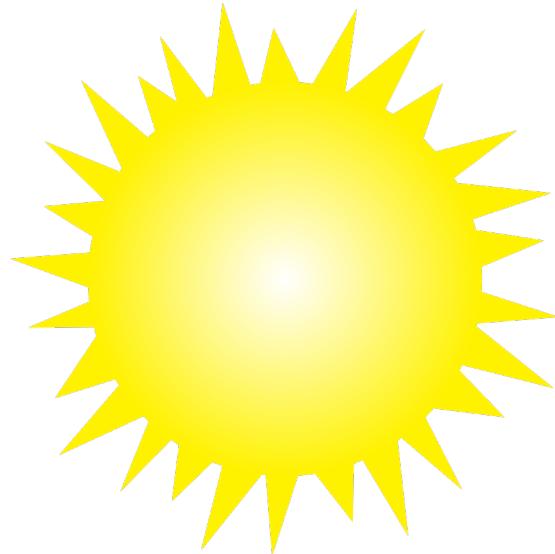


Then



Conditionals Part 2

If



Then



Otherwise



Conditionals in Python

```
shape = input("Should we fly a circle or a square?")
if (shape == "circle"):
    print("Let's fly in a circle!")
elif (shape == "square"):
    print("Let's fly in a square")
else:
    print("Shape %s is unknown. Help me learn how to fly it!" % shape)
```

Notice the use of colons here.

Your turn!

Ask the user how many sides they want to their polygon

Give them an error if it is < 3 or > 8

For the afternoon challenge, have your program actually fly the shape that the user specified!

For today, we will move on quickly to loops

Looping: While loop

- While teeth are not clean
 - Brush teeth
- While we have not reached the target
 - Fly forward



Quick While Loop Practice

- Syntax in python is:

```
count = 1
while (count <= 10):
    # do something
    print("Count is %d" % count)
```

Notice the use of colon here.

- Your quick task:

- Use a while loop to have your drone fly forward for 3 seconds total but in small movements. Do NOT just do fly_direct with a duration of 3!
- Max duration is 0.1
- Useful code

Looping: `for` loop

- Can we loop a specific number of times?
- You can use while but `for` loops are designed for this
- Python syntax

```
for side in range(1,4):  
    # fly forward  
    # turn 60 degrees
```

- One key thing to remember for python: numbers are inclusive on the lower end but exclusive on the higher end
- `range(1,4)` gives you numbers 1, 2, and 3 but NOT 4!

Looping: `for` loop

Example `for` loop:

```
for side in range(1,4):
    print("Flying side %d" % side)
```

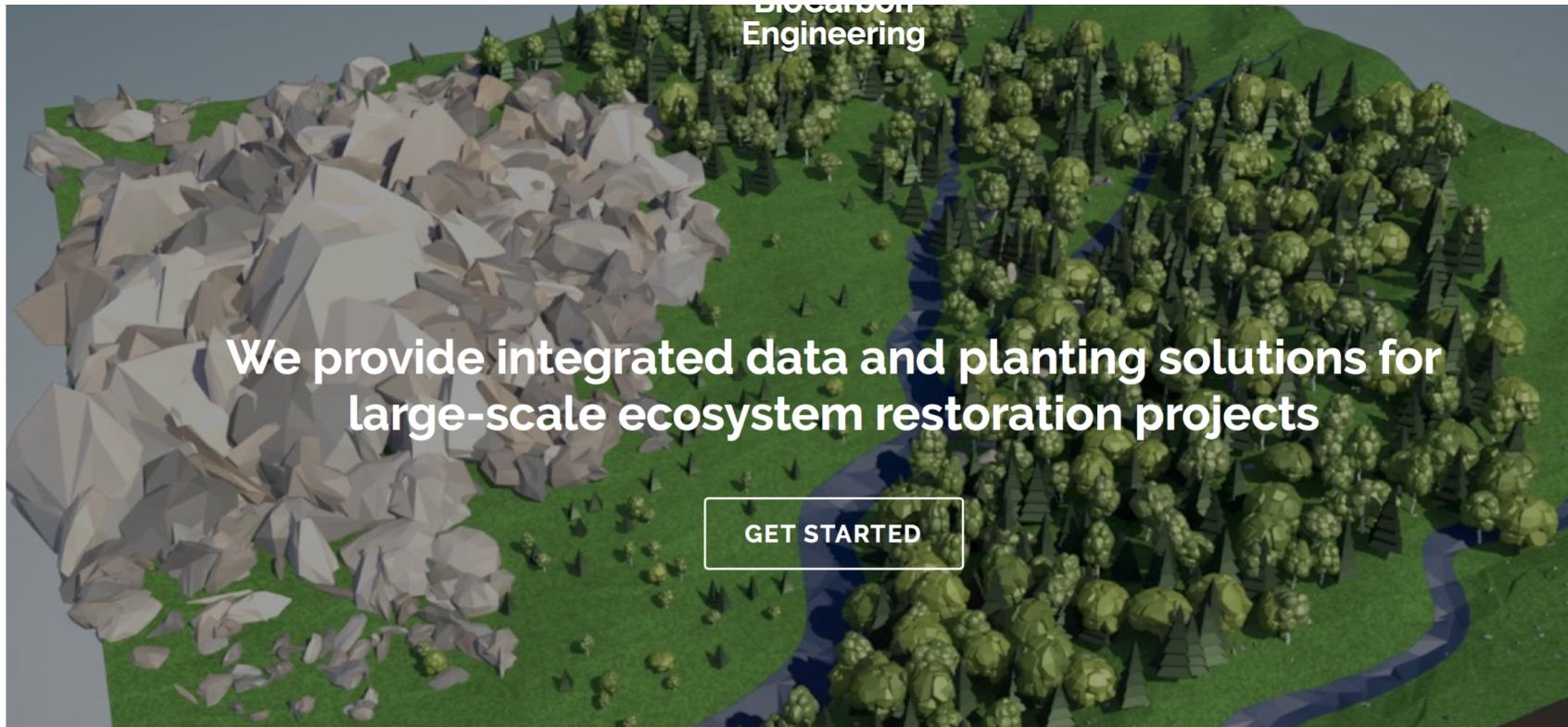
- How can you draw a triangle using a drone?

- For side = 1 to 3
 - Fly forward
 - Turn 60 degrees

```
for side in range(1,4):
    # fly forward
    # turn 60 degrees
```

Your turn!

Modify your conditional programs to actually fly the triangle and squares (or as many as you want, if you do it efficiently they are all in one loop) using a loop!



Planting New Trees

<https://www.youtube.com/watch?v=JcJ7vLwtSIM>

Sensors

- Mambo sensors
 - Flying state
 - Battery
 - Speed in x, y, and z (m/s)
 - Timestamp is in milliseconds!
 - Altitude (this is REALLY noisy)
 - Estimated orientation
 - All sensors updated at 2Hz

Using the Sensors: Flying state

- Flying state
`mambo.sensors.flying_state`
- Possible flying states

- “motor_ramping”
- “landed”
- “takingoff”
- “hovering”
- “flying”
- “landing”
- “emergency”
- “init”

Using the Sensors: Speed

- Speed in the forward direction:
`mambo.sensors.speed_x`
- Units of meters/second
- Average speed since the last update in speed
- Ignore y and z speeds for today

Using the Sensors: Timestep of the updates

- Timestep
 - `mambo.sensors.speed_ts`
- Measured in **milliseconds**
- Converting milliseconds to seconds:
 - `timestep_in_seconds = speed_ts / 1000`

Using the Sensors: Battery

- Battery
 - `mambo.sensors.battery`
- Initialized to 100%
- Becomes accurate after a few seconds
- Do NOT believe the initial 100%! Only believe it after a few seconds
- If your mambo eyes are blinking red, your batteries need to be charged!

Estimating Distance

- If I drive in my car at 55 mph for 1 hour, how far do I go?
- $Distance = rate * time$
- $Total\ distance = sum\ over\ all\ times\ (rate\ at\ time\ t\ * time\ spent\ at\ that\ speed)$

Using Sensors: Tying this all together

Activity: Fly the drone a user specified distance

Step 1: Ask the user how far you should fly (User Input)

Step 2: Create your mambo object and connect to it **after** asking the user

Step 3: Hover until the sensors are ready (Check function sheet for help)

Step 4: Estimate the distance flown using the sensors!

- Distance flown = distance flown already + new distance flown
- New distance flown = speed over the last x seconds * x seconds
- **Mambo: this will be more accurate for lower speeds**

Step 5: Fly until the distance meets or exceeds the distance the user specifies (While loop)

Step 6: Land and disconnect

Step 1: User input

- How do we ask the user how far we want to go?
 - `answer = input("How far do you want to fly?")`
- How do you save the result as a float and not a string (the default)
 - `distance = float(answer)`

Step 2: Create your mambo object and connect to it

- You can re-use your code from your earlier projects
- Create the mambo object, set the parameters for maximum speed, tell it to use flat_trim, and take off

```
from pymavlink import mavutil

# create the drone object and connect
drone = Mambo(use_wifi=True)
drone.connect(num_retries=3)

# maximum speeds - DO NOT CHANGE
drone.set_max_tilt(10)
drone.set_max_vertical_speed(1)

drone.flat_trim()

# takeoff
drone.safe_takeoff(5)
```

Step 3: Hover until the speed sensor is sending data

- The speed sensor does not send data right away
- Write a small (2 lines!) while loop that hovers as long as `mambo.sensors.speed_ts` is 0

Check function sheet for help!

while (condition):

check sensor

Step 4: Initialize your variables

- **timestep** – timestep from the last time you sensed speed
 - Mambo: approximately every 500 ms (but not always exactly equal)
 - Bebop: approximately every 100 ms
 - You need to track this for your calculation!
- **estimatedDistance** - Estimated distance you have traveled so far
 - This will be updated each time you get a new sensor reading
 - Initialize it to 0

Step 5: Fly until you exceed the user's distance

- While loop
 - What is the condition?
- Inside the loop:
 - Fly a SHORT amount (less than 0.5 seconds)
 - Sleep briefly (check function sheet for help)
 - Check the sensor values
 - Calculate how much time has passed
(`mambo.sensors.speed_ts`)
(~100 ms in bebop)
 - Compute how far you have moved
 - Distance = rate * time
 - `mambo.sensors.speed_x`
 - `bebop.sensors.sensors_dict["SpeedChanged_SpeedX"]`
 - **Remember time is in milliseconds!**

Step 6: Ending the loop

- You flew the distance!
- Land
- Disconnect

Code we typed in the workshop for sensors

```
from pyparrot.Minidrone import Mambo

# ask the user how far to fly
answer = input("How far do you want to fly in meters? ")
distance = float(answer)
print(distance)

# create the drone object and connect
drone = Mambo(use_wifi=True)
drone.connect(num_retries=3)

# maximum speeds - DO NOT CHANGE
drone.set_max_tilt(10)
drone.set_max_vertical_speed(1)

drone.flat_trim()

# takeoff
drone.safe_takeoff(5)

# hover while speed sensor is NOT working
while (drone.sensors.speed_ts == 0):
    drone.smart_sleep(0.1)

total_distance = 0
last_timestep = drone.sensors.speed_ts

while (total_distance <= distance):
    # do something flying related
    drone.fly_direct(roll=0, pitch=10, yaw=0, vertical_movement=0, duration=0.1)

    # how much time has elapsed?
    # convert from milliseconds to seconds
    elapsed_time = (drone.sensors.speed_ts - last_timestep)/1000.0

    # distance = rate times time
    total_distance = total_distance + drone.sensors.speed_x * elapsed_time

    # update last time step
    last_timestep = drone.sensors.speed_ts

drone.safe_land(2)
drone.disconnect()
```



Your turn!

Use loops and sensors and user input to have your drone fly forward a specific number of meters that your user wants

Be safe and don't hit the walls at full speed!

Vision slides to be used for
advanced groups only

Saving the Swimmers

- https://www.youtube.com/watch?v=OTM_dsGZZpl



<https://www.bbc.com/news/world-australia-42731112>

Drone Vision

- Vision is the best sensor for both drones!
- FPV camera
 - Vision runs at 30 frames per second
 - Approximately $\frac{1}{2}$ second delay on high-end laptops
 - Up to 1 second on slower laptops
 - Mambo: forward facing
 - Bebop: camera can be rotated (up and down)
- Mambo has a ground facing camera
 - Photos can be snapped and downloaded with about 0.1-0.3 second delay
 - Resolution is poor

Processing Images

- Image processing is a very difficult task in general
- OpenCV provides a lot of built in image processing tools
- We will use the aruco marker library today

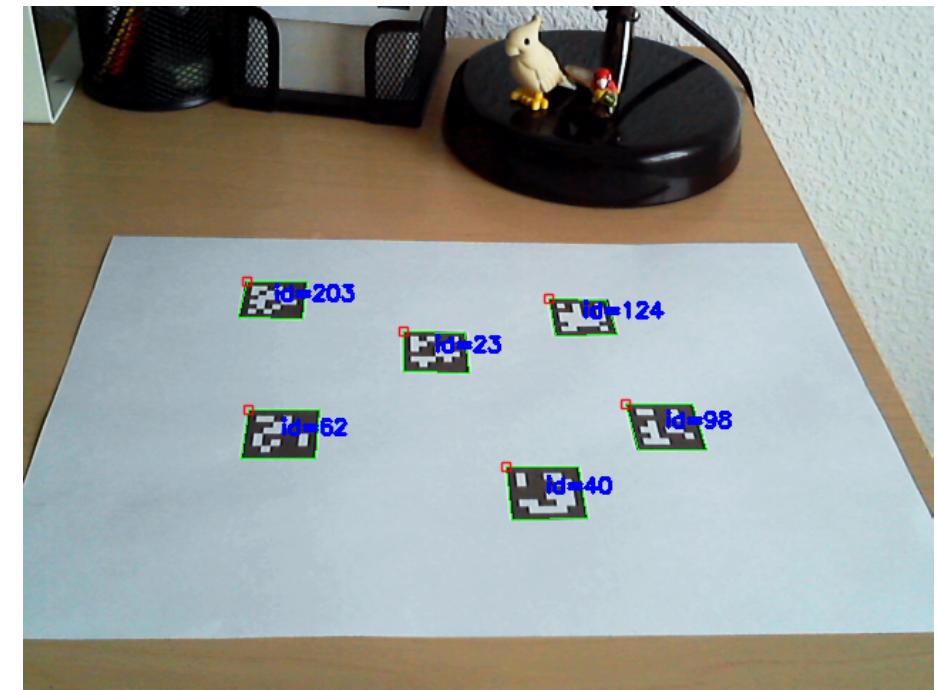
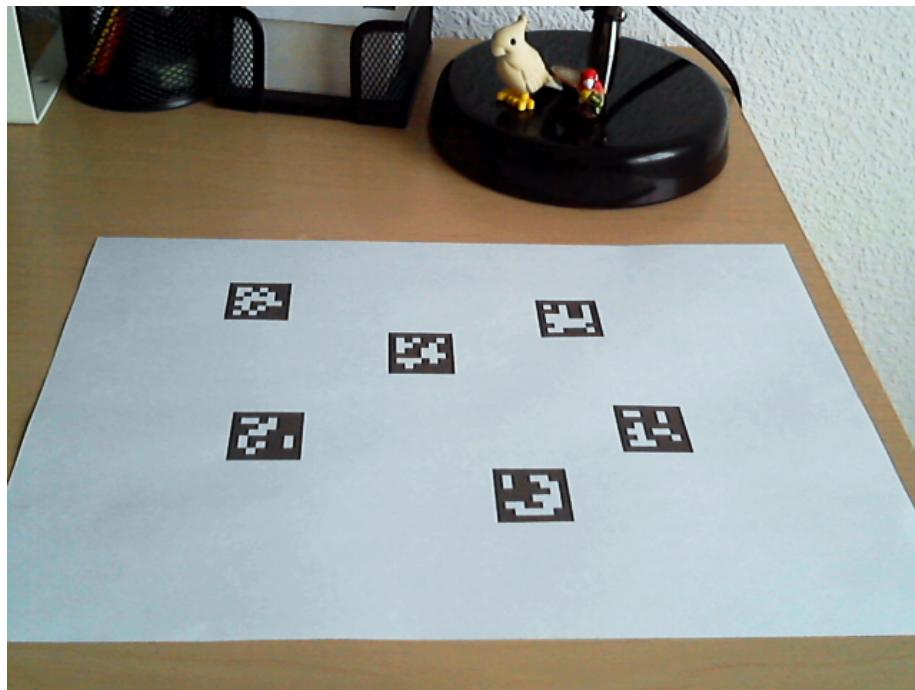


Image processing

- I have provided initial image processing code to you to simplify the process
- You are welcome to use this code for your team and modify it for the challenge

Starting code

```
def run_vision_code(droneVision, args):
    """
    Code that is run when the user presses "run" on the vision GUI

    :param droneVision: the drone vision object
    :param args: other arguments (first one is drone)
    :return:
    """

    # get the drone object out of the user arguments
    drone = args[0]

    # takeoff
    drone.safe_takeoff(5)
    drone.smart_sleep(3)

    ##### OK 2018 workshop - write the code for what you want your drone to do!
    # this is where you tell your drone to take off, fly, act according to the markers, land, etc
```

```
# cleanup the vision and the connection in this function at the end of your other code
drone.safe_land(5)
droneVision.close_video()

# disconnect nicely so we don't need a reboot
drone.disconnect()
```

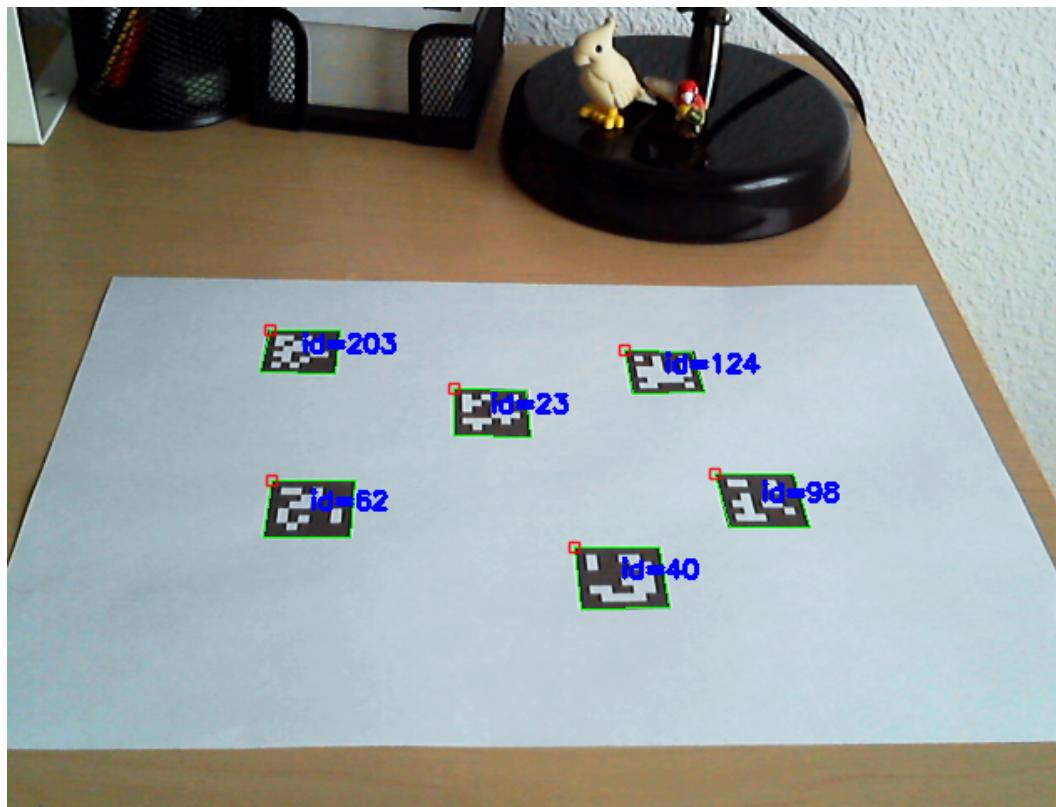
Aruco Marker Magic

- Detecting Aruco markers requires initialization

```
# parameters needed to make aruco vision work
aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
aruco_parameters = aruco.DetectorParameters_create()
```

Finding Aruco Markers

```
# find markers
corners, ids, rejectedImgPoints = aruco.detectMarkers(img, aruco_dict, parameters=aruco_parameters)
```



Aruco Markers

```
class ArucoMarker:  
    """  
        Computes and holds useful information about the aruco markers  
        all in a single object. Keeps the id, the centers, the corners, and the size.  
    """  
  
    def __init__(self, corners, id):  
        self.corners = corners  
        self.id = id  
  
        # calculate the center of the box  
        centers = np.mean(self.corners, axis=0)  
        self.center = (centers[0], centers[1])  
  
        # compute the size of the box  
        maxes = np.max(self.corners, axis=0)  
        mins = np.min(self.corners, axis=0)  
        self.size = (maxes[0] - mins[0]) * (maxes[1] - mins[1])
```

Creating Markers

```
def create_aruco_markers(corner_list, ids):
    """
    Makes the aruco_marker list from the data returned from aruco.detectMarkers
    :param corner_list: list of all the corners. Size is nx4x2 where n is the number of markers seen
    :param ids: ids for each marker
    :return: list of aruco_marker objects
    """

    if (ids is None):
        return list()

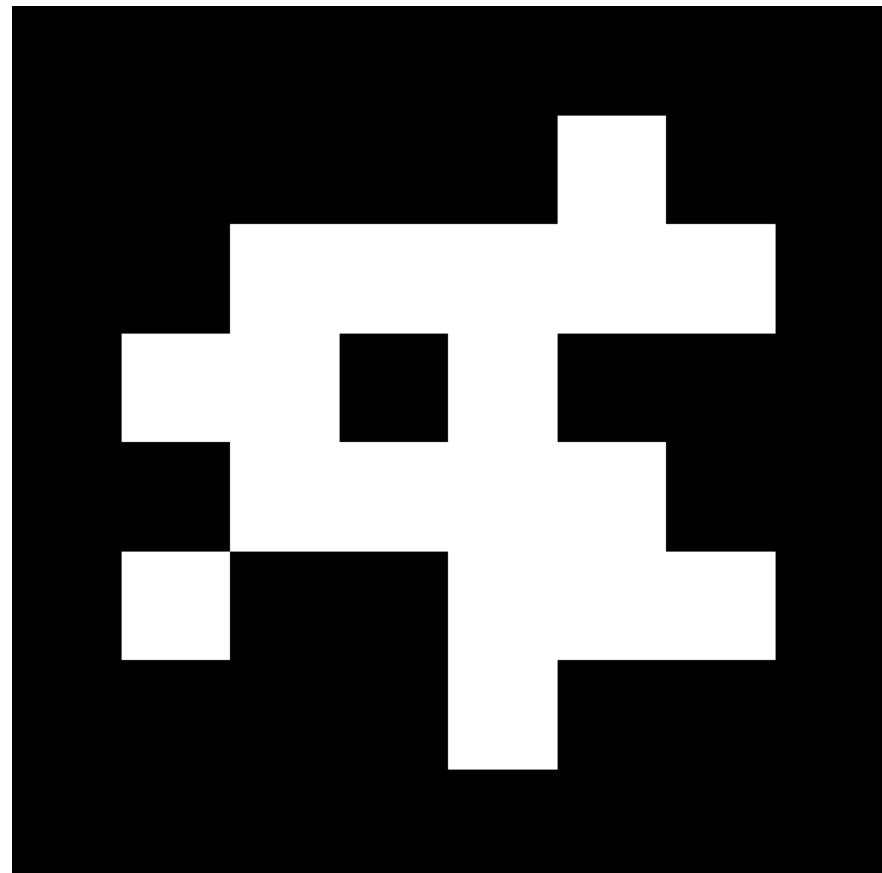
    markers = list()

    for counter, id in enumerate(ids):
        my_id = id[0]
        new_marker = ArucoMarker(np.squeeze(corner_list[counter]), my_id)
        markers.append(new_marker)

    return markers
```

Your Turn: First Vision Task

- Fill in the blank (where it says “OK Workshop put code here”)
- Have your drone hover until it sees Marker 16
- Marker 16 means flip
 - Make sure you send the flip command and then a sleep command for at least 2 seconds
- Land after you have flipped once
 - Repeated flips will cause a CRASH



Afternoon challenges

Regular challenges

1. Deliver blood to a single hospital
2. Deliver blood to multiple hospitals
3. Flying Art: N-sided Horizontal Polygons
4. Traversing obstacles (flying through hoops)

Advanced challenges

1. Flying Art: N-sided Vertical Polygons
2. Vision Challenge: Looking for a marker
3. Vision Challenge: Using vision to navigate obstacles and land precisely