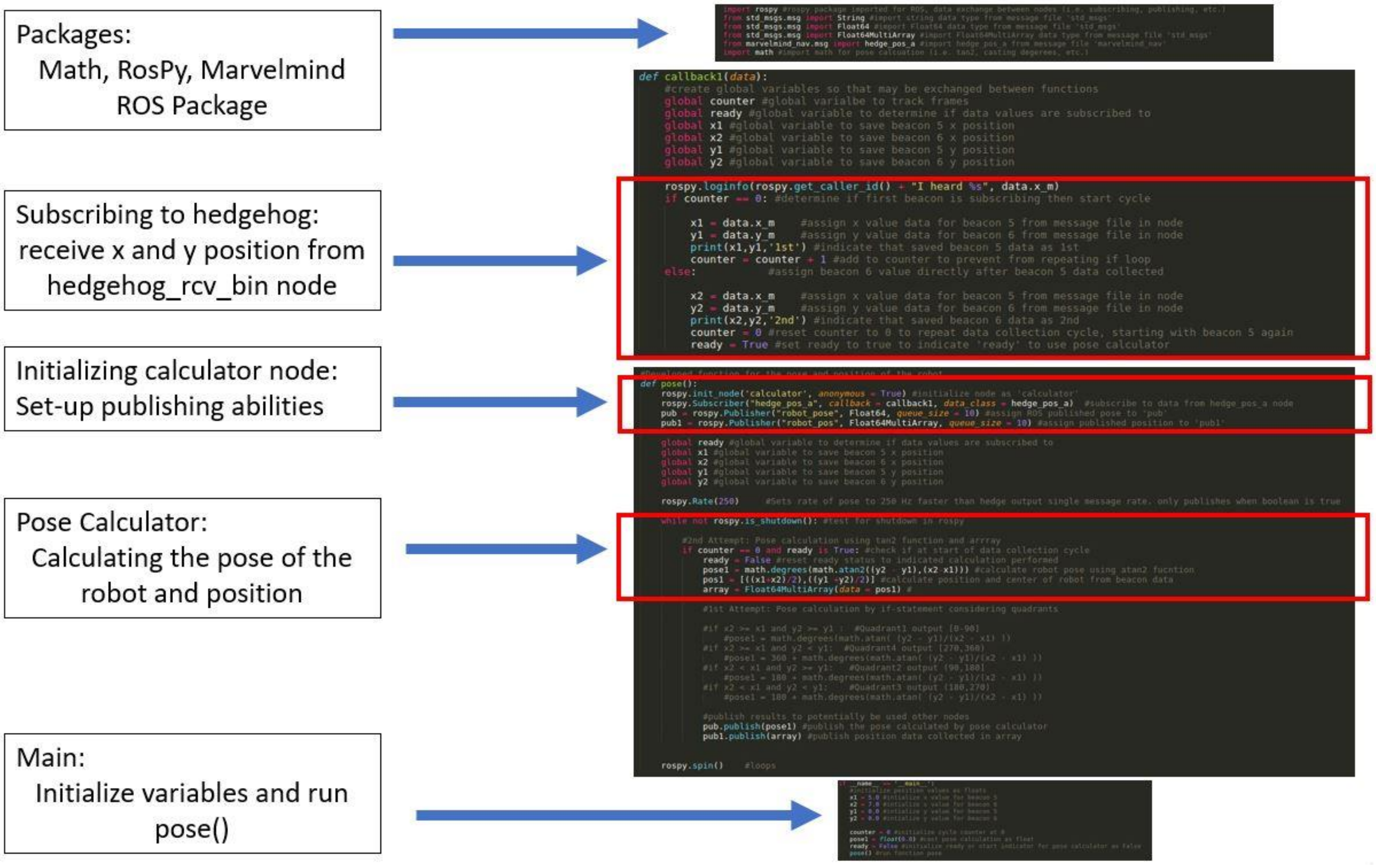
All the code written and contributed to for this project was written in Python. The starting files for the MarvelMind Indoor GPS System include files in C++ but since ROS is code agnostic, our pose calculator runs with the device set-up files. Our code structure can be broken down into three main sections aside from the calling the packages and the main loop. The three main sections include:

* Subscribing to the hedgehog (mobile) beacons

* Initializing our calculator node

* Pose and position calculation and output

The way the code works is such that two functions were created, one for the collection of data from the beacons, called 'callback1' and another for the pose and position calculation called 'pose'. The initialized variables are set up such that the program loops and first checks to see if the beacons are outputting data to the message file from the hedge_rcv_bin node. Then, once data indicates that the beacon is subscribing, a cycle starts in the subscribing to the hedgehog beacon section. The program uses a 'counter' variable to ensure that once one beacon's data is collected, the second beacon's data is collected after. The 'counter' is then reset to maintain this cycle. Every time a cycle is completed, the pose function is conditioned to run a position and pose calculation. The calculation is performed and outputted to two separate topics for each calculation.



Packages:
Math, RosPy, Marvelmind
ROS Package

```
import rospy #rospy package imported for ROS, data exchange between nodes (i.e. subscribing, publishing, etc.)
from std_msgs.msg import String #import string data type from message file 'std_msgs'
from std_msgs.msg import Float64 #import Float64 data type from message file 'std_msgs'
from std_msgs.msg import Float64MultiArray #import Float64MultiArray data type from message file 'std_msgs'
from marvelmind_nav.msg import hedge_pos_a #import hedge_pos_a from message file 'marvelmind_nav'
import math #import math for pose calcuation (i.e. tan2, casting degerees, etc.)
```

Subscribing to hedgehog:
receive x and y position from
hedgehog_rcv_bin node

```
def callback1(data):
    #create global variables so that may be exchanged between functions
    global counter #global varialbe to track frames
    global ready #global variable to determine if data values are subscribed to
    global x1 #global variable to save beacon 5 x position
    global x2 #global variable to save beacon 6 x position
    global y1 #global variable to save beacon 5 y position
    global y2 #global variable to save beacon 6 y position

    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.x_m)
    if counter == 0: #determine if first beacon is subscribing then start cycle

        x1 = data.x_m    #assign x value data for beacon 5 from message file in node
        y1 = data.y_m    #assign y value data for beacon 6 from message file in node
        print(x1,y1,'1st') #indicate that saved beacon 5 data as 1st
        counter = counter + 1 #add to counter to prevent from repeating if loop
    else:            #assign beacon 6 value directly after beacon 5 data collected

        x2 = data.x_m    #assign x value data for beacon 5 from message file in node
        y2 = data.y_m    #assign y value data for beacon 6 from message file in node
        print(x2,y2,'2nd') #indicate that saved beacon 6 data as 2nd
        counter = 0 #reset counter to 0 to repeat data collection cycle, starting with beacon 5 again
        ready = True #set ready to true to indicate 'ready' to use pose calculator
```

Initializing calculator node:
Set-up publishing abilities

```
#Developed function for the pose and position of the robot
def pose():
    rospy.init_node('calculator', anonymous = True) #initialize node as 'calculator'
    rospy.Subscriber("hedge_pos_a", callback = callback1, data_class = hedge_pos_a)  #subscribe to data from hedge_pos_a node
    pub = rospy.Publisher("robot_pose", Float64, queue_size = 10) #assign ROS published pose to 'pub'
    pub1 = rospy.Publisher("robot_pos", Float64MultiArray, queue_size = 10) #assign published position to 'pub1'

    global ready #global variable to determine if data values are subscribed to
    global x1 #global variable to save beacon 5 x position
    global x2 #global variable to save beacon 6 x position
    global y1 #global variable to save beacon 5 y position
    global y2 #global variable to save beacon 6 y position

    rospy.Rate(250)     #Sets rate of pose to 250 Hz faster than hedge output single message rate. only publishes when boolean is true
```

Pose Calculator:
Calculating the pose of the
robot and position

```
    while not rospy.is_shutdown(): #test for shutdown in rospy

        #2nd Attempt: Pose calculation using tan2 function and arrray
        if counter == 0 and ready is True: #check if at start of data collection cycle
            ready = False #reset ready status to indicated calculation performed
            pose1 = math.degrees(math.atan2((y2 - y1),(x2-x1))) #calculate robot pose using atan2 fucntion
            pos1 = [((x1+x2)/2),((y1+y2)/2)] #calculate position and center of robot from beacon data
            array = Float64MultiArray(data = pos1) #

            #1st Attempt: Pose calculation by if-statement considering quadrants

            #if x2 >= x1 and y2 >= y1 :  #Quadrant1 output [0-90]
                #pose1 = math.degrees(math.atan( (y2 - y1)/(x2 - x1) ))
            #if x2 >= x1 and y2 < y1:  #Quadrant4 output [270,360)
                #pose1 = 360 + math.degrees(math.atan( (y2 - y1)/(x2 - x1) ))
            #if x2 < x1 and y2 >= y1:   #Quadrant2 output (90,180]
                #pose1 = 180 + math.degrees(math.atan( (y2 - y1)/(x2 - x1) ))
            #if x2 < x1 and y2 < y1:    #Quadrant3 output (180,270)
                #pose1 = 180 + math.degrees(math.atan( (y2 - y1)/(x2 - x1) ))

            #publish results to potentially be used other nodes
            pub.publish(pose1) #publish the pose calculated by pose calculator
            pub1.publish(array) #publish position data collected in array

    rospy.spin()    #loops
```

Main:
Initialize variables and run
pose()

```
if __name__ == '__main__':
    #initialize position values as floats
    x1 = 5.0 #initialize x value for beacon 5
    x2 = 7.0 #initialize x value for beacon 6
    y1 = 0.0 #initialize y value for beacon 5
    y2 = 0.0 #initialize y value for beacon 6

    counter = 0 #initialize cycle counter at 0
    pose1 = float(0.0) #cast pose calculation as float
    ready = False #initialize ready as start indicator for pose calculator as false
    pose() #run function pose
```

An image of the code breakdown is provided. The code cycles as described and the three main sections are highlighted in the red boxes.