Appendix C – GUI Code

```
#Robotic Football Positioning System GUI
import math
from mpmath import *
from tkinter import *
import paho.mqtt.client as mqtt
#------------------------------------------------------------------------------
#------------------------------------------------------------------------------
#Definitions
global scale, width, length, robot_size, n, w, s, e, center_x, center_y, north,
       start_x, start_y, start_ang, final_x, final_y, final_ang
scale = 2
width = 354*scale #1128 inches in typical field
length = 347*scale #600 inches in typical field
robot_size = 16*scale #16 inches for typical robot
n = 10 #50
w = 10
s = length + n
e = width + w
center_x = e/2
center_y = s/2
north = radians(37)
start_x = center_x
start_y = center_y
start_ang = -pi/2 - north
start_x0 = (robot_size/2)*cos(start_ang) + start_x
start_y0 = (robot_size/2)*sin(start_ang) + start_y
final_x = 0
final_y = 0
final_ang = 0
final_x0 = 0
final_y0 = 0
global p1x, p1y, p2x, p2y, p3x, p3y, p4x, p4y, x_dir, y_dir, z_dir
p1x = 0 #w
p1y = 0 #n
p2x = width #e
p2y = 0 #n
p3x = width #e
p3y = length #s
p4x = 0 #w
p4y = length #s
x_dir = 0
y_dir = 0
z_dir = 0
global d1, d2, d3, d4, d1_array, d2_array, d3_array, d4_array, ack_array, a
d1 = 0
d2 = 0
d3 = 0
d4 = 0
d1_array = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
d2_array = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
d3_array = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
d4_array = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
```

```
ack_array = [0,0,0,0]
a = 0
global offset, x_offset, y_offset, z_offset
offset = 30 #calculated 3 May 2017
x_offset = 200
y_offset = 3000
z_offset = -600
#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
#Initialize GUI
global robot, arrow, resp_1, resp_2, resp_3, resp_4, arc_1, arc_2, arc_3, arc_4
gui = Tk()
gui.title("Robotic Football Positioning System")
gui.geometry('+{0}+{1}'.format(-7, -18))
gui.geometry('{0}x{1}'.format(width+w+10, length+n+10))
gui.config(bg="navy")
#Create title on GUI
#label = Label(gui, text="Robotic Football Positioning System", font="Times 24
bold", fg="gold", bg="navy")
#label.place(x=0, y=0)
#create playing field (responders should form corners)
field = Canvas(gui, highlightthickness=0, bg="gold")
field.place(x=w, y=n, width=width, height=length)
#create frames for responders
resp_1 = Canvas(gui, bg="green", highlightthickness=1)
resp_2 = Canvas(gui, bg="green", highlightthickness=1)
resp_3 = Canvas(gui, bg="green", highlightthickness=1)
resp_4 = Canvas(gui, bg="green", highlightthickness=1)
resp_1.place(x=w-10, y=n-10, width=20, height=20)
resp_2.place(x=e-10, y=n-10, width=20, height=20)
resp_3.place(x=e-10, y=s-10, width=20, height=20)
resp_4.place(x=w-10, y=s-10, width=20, height=20)
#create arcs for distance visualization
arc_1 = field.create_arc(p1x-60, p1y-60, p1x+60, p1y+60, start=270, extent=90,
style="arc", width=1)
arc_2 = field.create_arc(p2x-60, p2y-60, p2x+60, p2y+60, start=180, extent=90,
style="arc", width=1)
arc_3 = field.create_arc(p3x-60, p3y-60, p3x+60, p3y+60, start=90, extent=90,
style="arc", width=1)
arc_4 = field.create_arc(p4x-60, p4y-60, p4x+60, p4y+60, start=0, extent=90,
style="arc", width=1)
#create robot and orientation pointer
robot = field.create_oval(start_x-(robot_size/2), start_y-(robot_size/2),
start_x+(robot_size/2), start_y+(robot_size/2), fill="silver")
arrow = field.create_line(start_x, start_y, start_x0, start_y0, width=3,
                          arrow="last")

demo_x1 = field.create_line(p4x+(144*scale), p4y-(40*scale), p4x+(144*scale),
p4y-(60*scale), width=2, fill="orange")
demo_y1 = field.create_line(p4x+(134*scale), p4y-(50*scale), p4x+(154*scale),
p4y-(50*scale), width=2, fill="orange")
demo_x2 = field.create_line(p3x-(106*scale), p3y-(107*scale), p3x-(106*scale),
p3y-(127*scale), width=2, fill="green")
demo_y2 = field.create_line(p3x-(96*scale), p3y-(117*scale), p3x-(116*scale),
p3y-(117*scale), width=2, fill="green")
demo_x3 = field.create_line(p4x+(91*scale), p4y-(182*scale), p4x+(91*scale), p4y-
(202*scale), width=2, fill="blue")
```

```
demo_y3 = field.create_line(p4x+(81*scale), p4y-(192*scale), p4x+(101*scale),
p4y-(192*scale), width=2, fill="blue")
#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
#DEFINE FUNCTIONS
def on_connect(client, userdata, flags, rc):
    #Subscribing in on_connect() means reconnecting will renew subscriptions
    #Subscribe to MQTT channels
    client.subscribe("R1")
    client.subscribe("R2")
    client.subscribe("R3")
    client.subscribe("R4")
    client.subscribe("RX")
    client.subscribe("RY")
    client.subscribe("RZ")
    client.subscribe("Error")
    #Determine what to do with on_message based on MQTT channel
    client.message_callback_add("R1", on_message_1)
    client.message_callback_add("R2", on_message_2)
    client.message_callback_add("R3", on_message_3)
    client.message_callback_add("R4", on_message_4)
    client.message_callback_add("RX", on_message_X)
    client.message_callback_add("RY", on_message_Y)
    client.message_callback_add("RZ", on_message_Z)
    client.message_callback_add("Error", on_message_E)
#-------------------------------------------------------------------------------
def average(array):
    #Take average of an array
    return sum(array)/float(len(array))
def get_array(array, new_entry):
    #Get rid of oldest value and shift array with new value input
    del array[-1]
    array.insert(0, new_entry)
    return array
def Trilateration(x1, y1, x2, y2, x3, y3, r1, r2, r3):
    #Find the point where three distances intersect (using two circles and a ray)
    global width, length
    x_out = []
    y_out = []
    xd = x2 - x1
    yd = y2 - y1
    dist = math.sqrt(xd**2 + yd**2)
    k = (dist**2 + r1**2 - r2**2)/(2*dist)
    try:
        a = x1 + xd*k/dist + (yd/dist)*math.sqrt(r1**2 - k**2)
        b = y1 + yd*k/dist - (xd/dist)*math.sqrt(r1**2 - k**2)
        c = x1 + xd*k/dist - (yd/dist)*math.sqrt(r1**2 - k**2)
        d = y1 + yd*k/dist + (xd/dist)*math.sqrt(r1**2 - k**2)
        ad = x3 - a
        bd = y3 - b
        cd = x3 - c
        dd = y3 - d
        u = math.sqrt(ad**2 + bd**2)
        v = math.sqrt(cd**2 + dd**2)
        if (u>r3-12 and u<r3+12):
            x_out.insert(0, a)
            y_out.insert(0, b)
```

```
        elif (v>r3-12 and v<r3+12):
            x_out.insert(0, c)
            y_out.insert(0, d)
        if x_out[0]<0:
            x_out[0] = -x_out[0]
            if y_out<0:
                y_out[0] = -y_out[0]
            elif y_out>length:
                y_out[0] = y_out[0] - length
        elif x_out[0]>width:
            x_out[0] = x_out[0] - width
            if y_out<0:
                y_out[0] = -y_out[0]
            elif y_out>length:
                y_out[0] = y_out[0] - length
        return x_out, y_out
    except:
            return [], []
def Position():
    #Find position of robot on field, based on average from trilateration of all
responders
    global d1, d2, d3, d4, p1x, p1y, p2x, p2y, p3x, p3y, p4x, p4y, width, length,
final_x, final_y, final_ang
    global robot
    x_123, y_123 = Trilateration(p1x, p1y, p2x, p2y, p3x, p3y, d1, d2, d3)
    x_132, y_132 = Trilateration(p1x, p1y, p3x, p3y, p2x, p2y, d1, d3, d2)
    x_124, y_124 = Trilateration(p1x, p1y, p2x, p2y, p4x, p4y, d1, d2, d4)
    x_142, y_142 = Trilateration(p1x, p1y, p4x, p4y, p2x, p2y, d1, d4, d2)
    x_134, y_134 = Trilateration(p1x, p1y, p3x, p3y, p4x, p4y, d1, d3, d4)
    x_143, y_143 = Trilateration(p1x, p1y, p4x, p4y, p3x, p3y, d1, d4, d3)
    x_231, y_231 = Trilateration(p2x, p2y, p3x, p3y, p1x, p1y, d2, d3, d1)
    x_241, y_241 = Trilateration(p2x, p2y, p4x, p4y, p1x, p1y, d2, d4, d1)
    x_234, y_234 = Trilateration(p2x, p2y, p3x, p3y, p4x, p4y, d2, d3, d4)
    x_243, y_243 = Trilateration(p2x, p2y, p4x, p4y, p3x, p3y, d2, d4, d3)
    x_341, y_341 = Trilateration(p3x, p3y, p4x, p4y, p1x, p1y, d3, d4, d1)
    x_342, y_342 = Trilateration(p3x, p3y, p4x, p4y, p2x, p2y, d3, d4, d2)
    x_pos = x_123+x_132+x_124+x_142+x_134+x_143+x_231+x_241+x_234+x_243+x_341+x_342
    y_pos = y_123+y_132+y_124+y_142+y_134+y_143+y_231+y_241+y_234+y_243+y_341+y_342
    try:
        final_x = average(x_pos)
        final_y = average(y_pos)
        print("x: "+str(final_x)+", "+str(final_y))
        field.coords(robot, final_x-(robot_size/2), final_y-(robot_size/2),
                     final_x+(robot_size/2), final_y+(robot_size/2))
    except:
            pass
def Orientation(a_dir, b_dir):
    #Find the direction the robot is facing
    global north, final_x, final_y, final_ang
    global arrow
    try:
        if (a_dir!=0 and b_dir!=0):
            ang = atan(a_dir, b_dir)
        final_ang = ang - north
        if final_angle < -pi:
            final_ang = final_ang + 2*pi
        elif final_and > pi:
```

```
            final_ang = final_ang - 2*pi
        final_deg = degrees(final_ang)
        final_x0 = (robot_size/2)*cos(final_ang) + final_x
        final_y0 = (robot_size/2)*sin(final_ang) + final_y
        #print(final_x0, final_y0)
        field.coords(arrow, final_x, final_y, final_x0, final_y0)
    except:
            pass
def ack_timing(ack_array):
    #Change responder visual if there is no response
    if ack_array[0]==0:
        resp_1.config(resp_1, bg="red")
    else:
        resp_1.config(resp_1, bg="green")
    if ack_array[1]==0:
        resp_2.config(resp_2, bg="red")
    else:
        resp_2.config(resp_2, bg="green")
    if ack_array[2]==0:
        resp_3.config(resp_3, bg="red")
    else:
        resp_3.config(resp_3, bg="green")
    if ack_array[3]==0:
        resp_4.config(resp_4, bg="red")
    else:
        resp_4.config(resp_4, bg="green")
#------------------------------------------------------------------------------
#messages to R1-R4 should arrive 10-12 times a second
#messages to RX, RY, and RZ should only arrive every second and half
#messages to Error only ever couple seconds
def on_message_1(client, userdata, msg):
    #Calculates average distance from responder 1 and sets arc
    #Runs the positioning function
    global d1, d1_array, offset, scale
    global arc_1
    try:
        d1_new = int(msg.payload)
        if d1_new!=0:
            get_array(d1_array, d1_new)
            if d1_array[len(d1_array)-1] != 0:
                d1 = (average(d1_array) - offset)*scale
                #print("D1: "+str(d1))
                Position()
                field.coords(arc_1, -d1, -d1, d1, d1)
    except:
            pass
def on_message_2(client, userdata, msg):
    #Calculates average distance from responder 2 and sets arc
    global d2, d2_array, offset, scale
    global arc_2
    try:
        d2_new = int(msg.payload)
        if d2_new!=0:
            get_array(d2_array,d2_new)
            if d2_array[len(d2_array)-1] != 0:
                d2 = (average(d2_array) - offset)*scale
                #print("D2: "+str(d2))
```

```
                Position()
                field.coords(arc_2, width-d2, -d2, width+d2, d2)
    except:
            pass
def on_message_3(client, userdata, msg):
    #Calculates average distance from responder 3 and sets arc
    global d3, d3_array, offset, scale
    global arc_3
    try:
        d3_new = int(msg.payload)
        if d3_new!=0:
            get_array(d3_array,d3_new)
            if d3_array[len(d3_array)-1] != 0:
                d3 = (average(d3_array) - offset)*scale
                #print("D3: "+str(d3))
                Position()
                field.coords(arc_3, width-d3, length-d3, width+d3, length+d3)
    except:
            pass
def on_message_4(client, userdata, msg):
    #Calculates average distance from responder 4 and sets arc
    global d4, d4_array, offset, scale
    global arc_4
    try:
        d4_new = int(msg.payload)
        if d4_new!=0:
            get_array(d4_array,d4_new)
            if d4_array[len(d4_array)-1] != 0:
                d4 = (average(d4_array) - offset)*scale
                #print("D4: "+str(d4))
                Position()
                field.coords(arc_4, -d4, length-d4, d4, length+d4)
    except:
            pass
def on_message_X(client, userdata, msg):
    #Calculates X coordinate based on input
    global x_dir, y_dir, z_dir, x_offset, scale
    global arrow
    try:
        x_dir = (float(msg.payload)-x_offset)
        #print("X: "+str(x_dir))
        #Orientation(x_dir, y_dir)
        #Orientation(y_dir, x_dir)
        #Orientation(x_dir, z_dir)
        #Orientation(z_dir, z_dir)
    except:
            pass
def on_message_Y(client, userdata, msg):
    #Calculates Y coordinate based on input
    global x_dir, y_dir, z_dir, y_offset, scale
    global arrow
    try:
        y_dir = (float(msg.payload)-y_offset)
        #print("Y: "+str(y_dir))
        #Orientation(y_dir, x_dir)
        #Orientation(x_dir, y_dir)
        #Orientation(y_dir, z_dir)
```

```
        #Orientation(z_dir, y_dir)
    except:
            pass
def on_message_Z(client, userdata, msg):
    #Calculate Z coordinate based on input
    global x_dir, y_dir, z_dir, z_offset, scale
    global arrow
    try:
        #z_dir = (float(msg.payload)-z_offset)
        print("Z: "+str(z_dir))
        #Orientation(z_dir, x_dir)
        #Orientation(x_dir, z_dir)
        #Orientation(z_dir, y_dir)
        #Orientation(y_dir, z_dir)
    except:
            pass
def on_message_E(client, userdata, msg):
    #Check responder acknowledgement messages and show if one has failed
    global ack_array, a
    if "1" in str(msg.payload):
        ack_array[0] = 1
        a += 1
    elif "2" in str(msg.payload):
        ack_array[1] = 1
        a += 1
    elif "3" in str(msg.payload):
        ack_array[2] = 1
        a += 1
    elif "4" in str(msg.payload):
        ack_array[3] = 1
        a += 1
    if a%12==0:
        ack_timing(ack_array)
#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
#RUN PROGRAM
client = mqtt.Client()
host = "192.168.1.136" #Rasberry Pi IP address

client.on_connect = on_connect
client.on_message_1 = on_message_1
client.on_message_2 = on_message_2
client.on_message_3 = on_message_3
client.on_message_4 = on_message_4
client.on_message_X = on_message_X
client.on_message_Y = on_message_Y
client.on_message_Z = on_message_Z
client.connect(host, 1883, 60)
#-------------------------------------------------------------------------------
def task():
    gui.update()
    client.loop(.1)
    #Position()

while 1:
    task()
```