pathing()

if (arena.loop\_count == 0) //Running in blind mode

if(wall\_hit == false) //hasnt hit the wall yet

if(Timer.started == false) //If timer hasnt start ->start the timer

Timer.start\_time = millis(); //record the time robot start moving

Timer.started == true;

return pathing;

Else if(Timer.started == true) //If robot has started moving

Timer.elapsed = millis() - Timer.start\_time; //record total elapsed time since start of movement

forward(); //Move forward

return pathing;

if(wall\_hit == true) //If wall has been reached

wall\_hit = false; //Reset status flag

elapsed\_time = millis() - Timer.start\_time(); //Find time takes to reach wall since start of movement

Timer.started = false; //reset clock flag;

stop(); //Stop robot to prepare for rotation and alignment

if (corner\_count == 4) //If the first full loop of the arena has been completed

int temp;

arena.width = (arena.size[1] + arena.size[3])/2; //Find the averaged value of an undetermined dimension of the arena

arena.length = (arena.size[2] + arena.size[4])/2; //Find another averaged value of an undetermined dimension of the arena

if(arena.width > arena.length) // The longer averaged value would be the length and the other one would be the width of the arena

temp = arena.width;

arena.width = arena.length;

arena.length = temp;

arena.corner\_count = 0;

arena.loop\_count = 1;

called\_state = pathing; //Pass info about the state that called the rotation state to ensure accurate return point

return ROTATE\_90\_CW; //Rotate, align and repeat

else // If first full loop has not been completed

arena.size[arena.index] = average\_velocity \* elapsed\_time; //Calculate the distance of the previously travelled wall

corner\_count ++; //increment counters

arena.index ++;

called\_state = pathing; //Pass info about the state that called the rotation state to ensure accurate return point

return ROTATE\_90\_CW; //Rotate, align and repeat

Else if (arena.loop\_count > 0 && arena.loop\_count < 4) //Running when dimensions are known and the full arena has not been mapped

switch(arena.loop\_count) // Calculate the expected y distance from wall of robot

case 1:

expected\_y = ((arena.size[1] + arena.size[3])/2) - off\_set;

break;

case 2:

expected\_y = ((arena.size[2] + arena.size[4])/2) - off\_set;

break;

case 3:

expected\_y = ((arena.size[1] + arena.size[3])/2) - off\_set;

break;

case 4:

expected\_y = ((arena.size[2] + arena.size[4])/2) - off\_set;

break;

switch(arena.loop\_count) // Calculate the expected x distance from wall of robot

case 1:

expected\_x = ((arena.size[1] + arena.size[3])/2) - off\_set;

break;

case 2:

expected\_x = ((arena.size[2] + arena.size[4])/2) - off\_set;

break;

case 3:

expected\_x = ((arena.size[1] + arena.size[3])/2) - off\_set;

break;

case 4:

expected\_x = ((arena.size[2] + arena.size[4])/2) - off\_set;

break;

if(wall\_hit == false || current\_y < expected\_y) // If expected

if(initial\_y\_read == false && Timer.started == false)

Timer.start\_time = millis();

Timer.started = true;

initial\_y = sonar\_reading;

initial\_y\_read = true;

if(initial\_y\_read == true)

current\_y = initial\_y\_read - sonar\_reading;

Timer.started = false;

xPID.SetMode(AUTOMATIC); //turn on PID control for x direction

yPID.setMode(AUTOMATIC);

double xgap = abs(Long\_range\_IR\_readings - setpoint.x)

if (gap < 2)

xPID.SetTunings(consKp,consKi,consKd); // If little error then use conservative gains

else

xPID.SetTunings(aggKp,aggKi,aggKd); //If error large then use aggresive gains

double ygap = abs(current\_y - expected\_y)

if (gap < 2)

yPID.SetTunings(consKp,consKi,consKd); // If little error then use conservative gains

else

yPID.SetTunings(aggKp,aggKi,aggKd); //If error large then use aggresive gains

xPID.Compute();

yPID.Compute();

PID\_forward();